EL GRAN LIBRO DE ANDROID

JESÚS TOMÁS GIRONÉS
JAIME LLORET MAURI















El gran libro de Android

Jesús Tomás Gironés
Jaime Lloret Mauri

Marcombo

Índice general

Lista de siglas y acrónimos xvi

¿Cómo leer este libro? xviii

CAPÍTULO 1. Visión general y entorno de desarrollo 21

1.1. ¿Qué hace que Android sea especial? 22

1.2. Los orígenes 23

1.3. Comparativa con otras plataformas 24

1.4. Arquitectura de Android 26

1.4.1. El núcleo Linux 27

1.4.2. *Runtime* de Android 27

1.4.3. Librerías nativas 28

1.4.4. Entorno de aplicación 28

1.4.5. Aplicaciones 29

1.5. Instalación del entorno de desarrollo 29

1.5.1. Instalación de la máquina virtual Java 29

1.5.2. Instalación de Android Studio 30

1.5.3. Creación de un dispositivo virtual Android (AVD) 33

1.6. Las versiones de Android y niveles de API 36

1.6.1. Las primeras versiones 36

1.6.2. Cupcake 37

1.6.3. Donut 37

1.6.4. Éclair 37

1.6.5. Froyo 38

1.6.6. Gingerbread 38

1.6.7. Honeycomb 39

1.6.8. Ice Cream Sandwich 40

1.6.9. Jelly Bean 41

1.6.10. KitKat 41

1.6.11. Lollipop 42

1.6.12. Marshmallow 44

3.4.2. Uso del patrón Singleton .... [illegible]
3.5. Uso de la arquitectura Clean en Mis Lugares .... [illegible]
Capa de Modelo .... [illegible]
Capa de Datos .... [illegible]
Capa de Casos de Uso .... [illegible]
Capa de Presentación .... [illegible]
Organizando las clases en paquetes .... [illegible]
3.6. Creando actividades en Mis Lugares 162
3.6.1. Creando la actividad VistaLugarActivity 162
3.6.2. Creando la actividad EdicionLugarActivity 173
3.7. Creación y uso de iconos 173
3.8. Añadiendo preferencias de usuario 183
3.8.1. Organizando preferencias<opcional> 187
3.8.2. Cómo se almacenan las preferencias de usuario<opcional> 188
3.8.3. Accediendo a los valores de las preferencias 189
3.8.4. Verificar valores correctos<opcional> 190
3.9. Añadiendo una lista de puntaciones en Asteroides 191
3.10. Creación de listas con RecyclerView 193
Personalizar los datos a mostrar 194
Distribuir los elementos 195
3.11. Las intenciones 206
3.11.1. Añadiendo fotografías en Mis Lugares 214
CAPÍTULO 4. Gráficos en Android **223**
4.1. Clases para gráficos en Android 224
4.1.1. Canvas 224
4.1.2. Paint 227
Definición de colores 228
4.1.3. Path 229
4.1.4. Drawable 231
BitmapDrawable 232
VectorDrawable 233
GradientDrawable 237
TransitionDrawable 238

ShapeDrawable 238
AnimationDrawable 239
4.2. Creación de una vista en un fichero independiente 240
4.3. Creando la actividad principal de Asteroides 244
4.3.1. La clase Gráfico 245
4.3.2. La clase VistaJuego 248
4.3.3. Introduciendo la nave en VistaJuego 249
4.4. Representación de gráficos vectoriales en Asteroides 251
4.5. Animaciones 255
4.5.1. Animaciones de vistas 255
4.5.2. Animaciones de propiedades 258
CAPÍTULO 5. Hilos de ejecución, pantalla táctil y sensores **261**
5.1. Uso de hilos de ejecución (*threads*) 262
5.1.1. Introducción a los procesos e hilos de ejecución 262
5.1.2. Hilos de ejecución en Android 262
5.1.3. Creación de nuevos hilos con la clase Thread 265
5.1.4. Introduciendo movimiento en Asteroides 268
5.1.5. Ejecutar una tarea en un nuevo hilo con AsyncTask 271
5.1.6. Mostrar un cuadro de progreso en un AsyncTask 274
5.1.7. El método *get()* de AsyncTask 276
5.2. Manejando eventos de usuario 278
5.2.1. Escuchador de eventos de la clase View 278
5.2.2. Manejadores de eventos 280
5.3. El teclado 280
5.4. La pantalla táctil 283
5.4.1. Manejo de la pantalla táctil *multi-touch* 285
5.4.2. Manejo de la nave con la pantalla táctil 288
5.5. Los sensores 289
5.5.1. Un programa que muestra los sensores disponibles y sus valores en tiempo real 294
5.5.2. Utilización de los sensores en Asteroides 296
5.5.3. Restricciones al uso de sensores en segundo plano en Android 9 .. 298
5.6. Introduciendo un misil en Asteroides 298

CAPÍTULO 6. Multimedia y ciclo de vida de una actividad 305

6.1. Ciclo de vida de una actividad 306

6.1.1. ¿Qué proceso se elimina? 310

6.1.2. Guardando el estado de una actividad 312

6.2. Utilizando multimedia en Android 315

6.3. La vista VideoView 317

6.4. La clase MediaPlayer 319

6.4.1. Reproducción de audio con MediaPlayer 320

6.5. Un reproductor multimedia paso a paso 325

6.6. Introduciendo efectos de audio con SoundPool 327

6.7. Grabación de audio 330

CAPÍTULO 7. Seguridad y posicionamiento 335

7.1. Los tres pilares de la seguridad en Android 336

7.1.1. Ejecución en procesos independientes Linux 337

7.1.2. Firma digital de los apks 337

7.1.3. El esquema de permisos en Android 338

7.1.4. Permisos desde Android 6 Marshmallow 344

7.1.5. Permisos definidos por el programador en Android 350

7.1.6. Cambios relacionados con la privacidad en Android 9 354

7.1.7. Cambios relacionados con la privacidad en Android 10 354

7.2. Localización 355

7.2.1. Sistemas de geolocalización en dispositivos móviles 356

7.2.2. La API de localización de Android 356

7.2.3. Emulación del GPS con Android Studio 363

7.2.4. Estrategias para escoger un proveedor de localización 363

Usar siempre el mismo tipo de proveedor 364

El mejor proveedor según un determinado criterio 364

Usar los dos proveedores en paralelo 364

7.2.5. Límites de ubicación en segundo plano 372

7.3. Google Maps 373

7.3.1. Obtención de una clave Google Maps 374

CAPÍTULO 8. Servicios, notificaciones y receptores de anuncios 391

8.1. Introducción a los servicios en Android 392

8.1.1. Ciclo de vida de un servicio 393

8.1.2. Permisos 395

8.2. Un servicio para ejecución en segundo plano 395

8.2.1. El método onStartCommand() 398

8.3. Un servicio en un nuevo hilo con IntentService 399

8.3.1. La clase IntentService 402

8.4. Las notificaciones de la barra de estado 404

8.4.1. Configurando tipos de avisos en las notificaciones 408

Asociar un sonido 409

Añadiendo vibración 409

Añadiendo parpadeo de LED 409

8.4.2. Servicios en primer plano 410

8.5. Receptores de anuncios 412

8.5.1. Receptor de anuncios registrado en *AndroidManifest.xml* 413

8.5.2. Receptor de anuncios registrado por código 415

8.5.3. Arrancar una actividad en una nueva tarea desde un receptor de anuncio 420

8.5.4. Arrancar un servicio tras cargar el sistema operativo 422

8.6. Un receptor de anuncios como mecanismo de comunicación 423

8.6.1. Receptores de anuncios desde Android 8 425

8.6.2. Anuncios *broadcast* permanentes 425

8.7. Un servicio como mecanismo de comunicación entre aplicaciones 426

8.7.1. Crear la interfaz en AIDL 427

8.7.2. Implementar la interfaz 428

8.7.3. Publicar la interfaz en un servicio 429

8.7.4. Llamar a una interfaz remota 430

CAPÍTULO 9. Almacenamiento de datos 433

9.1. Alternativas para guardar datos permanentemente en Android 434

9.2. Añadiendo puntuaciones en Asteroides 435

9.2.1. Fragmentando los asteroides 437

9.3. Preferencias 438

9.4. Accediendo a ficheros 442

9.4.1. Sistema interno de ficheros 442

9.4.2. Sistema de almacenamiento externo .... 44[illegible]
Verificando acceso a la memoria externa .... 44[illegible]
Almacenando ficheros específicos de tu aplicación en el almacenamiento externo .... 44[illegible]
Almacenando ficheros compartidos en el almacenamiento externo 44[illegible]
Almacenamiento externo con varias unidades .... 44[illegible]
9.4.3. Acceder a un fichero de los recursos .... 45[illegible]
9.5. Trabajando con XML .... 45[illegible]
9.5.1. Procesando XML con SAX .... 45[illegible]
9.5.2. Procesando XML con DOM .... 45[illegible]
9.6. Trabajando con JSON .... 45[illegible]
9.6.1. Procesando JSON con la librería Gson .... 46[illegible]
9.6.2. Procesando JSON con la librería org.json .... 46[illegible]
9.7. Bases de datos con SQLite .... 46[illegible]
9.7.1. Los métodos *query()* y *rawQuery()* .... 46[illegible]
9.7.2. Uso de bases de datos en Mis Lugares 471
9.7.3. Adaptadores para bases de datos 474
Operaciones con bases de datos en Mis Lugares 483
9.7.4. Bases de datos relacionales 493
9.7.5. El método *onUpgrade* de la clase SQLiteOpenHelper 496
9.8. Content Provider 497
9.8.1. Conceptos básicos 498
El modelo de datos 498
Las URI 498
9.8.2. Acceder a la información de un ContentProvider 499
Leer información de un ContentProvider 500
Escribir información en un ContentProvider 502
Borrar y modificar elementos de un ContentProvider 503
9.8.3. Creación de un ContentProvider 504
Definir la estructura de almacenamiento del ContentProvider 504
Extendiendo la clase ContentProvider 505
Declarar el ContentProvider en *AndroidManifest.xml* 509
9.8.4. Acceso a PuntuacionesProvider desde Asteroides 510

CAPÍTULO 10. Internet: *sockets*, HTTP y servicios web **513**
10.1. Comunicaciones en Internet mediante *sockets* 514
10.1.1. La arquitectura cliente/servidor 514
10.1.2. ¿Qué es un *socket*? 514
Sockets stream (TCP) 515
Sockets datagram (UDP) 515
10.1.3. Un ejemplo de un cliente/servidor de ECHO 516
10.1.4. Un servidor por *sockets* para las puntuaciones 520
10.2. La web y el protocolo HTTP 524
10.2.1. El protocolo HTTP 524
10.2.2. Versión 1.0 del protocolo HTTP 526
10.2.3. Utilizando HTTP desde Android 527
10.2.4. Uso de HTTP con AsyncTask 532
10.3. La librería Volley 534
10.3.1. Descargar un String con Volley 534
10.3.2. Paso de parámetros con el método POST 537
10.3.3. Descargar imágenes con Volley 537
10.4. Servicios web 540
10.4.1. Alternativas en los servicios web 541
Servicios web basados en SOAP 541
Servicios web basados en REST 542
10.4.2. Acceso a servicios web de terceros 547
10.4.3. Un servicio web con Apache, PHP y MySQL 550
Utilizando el servicio web PHP desde Asteroides 555
Creación de un servicio web en un servidor de *hosting* 557
Utilizando AsyncTask de forma síncrona 561
10.4.4. Comparativa *sockets*/servicios web 564
ANEXO A. Diálogos de fecha y hora **565**
Clases para trabajar con fechas en Java 565
ANEXO B. *Fragments* **575**
ANEXO C. Referencia Java **591**
ANEXO D. Referencia de la clase View y sus descendientes **595**
ANEXO E. Sufijos utilizados en recursos alternativos **595**

Lista de siglas y acrónimos

AIDL	*Android Interface Definition Language*
API	*Application Programming Interface*
AVD	*Android Virtual Device*
ART	*Android RunTime*
CSS	*Cascading Style Sheets*
CORBA	*Common Object Request Broker Architecture*
CPU	*Central Processing Unit*
DOM	*Document Object Model*
DTD	*Document Type Definition*
FTP	*File Transfer Protocol*
GPU	*Graphic Processing Unit*
GPS	*Global Positioning System*
GSM	*Global System for Mobile communications*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
IMEI	*International Mobile Equipment Identity*
IMSI	*International Mobile Subscriber Identity*
IU	*Interfaz de Usuario*
JAR	*Java ARchive*
JDK	*Java Development Kit*
JRE	*Java Runtime Environment*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
MCC	*Mobile Country Code*
MNC	*Mobile Network Code*
MIME	*Multipurpose Internet Mail Extensions*
MTP	*Media Transfer Protocol*
NFC	*Near Field Communication*
NDK	*Native Development Kit*
OpenGL	*Open Graphic Library*

PCM	*Pulse-Code Modulation*
PDA	*Personal Digital Assistant*
PNG	*Portable Network Graphics*
PHP	*Hypertext Pre-processor*
PTP	*Picture Transfer Protocol*
RAM	*Random Access Memory*
REST	*Representational State Transfer*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Calls*
SAX	*Simple API for XML*
SD	*Secure Digital*
SDK	*Software Developers Kit*
SMS	*Short Message Service*
SIM	*Subscriber Identity Module*
SO	*Sistema Operativo*
SOA	*Service-Oriented Architecture*
SOAP	*Simple Object Access Protocol*
SQL	*Structured Query Language*
SVG	*Scalable Vector Graphics*
TCP	*Transmission Control Protocol*
UI	*User Interface*
URL	*Universal Resource Locator*
URI	*Uniform Resource Identifier*
USB	*Universal Serial Bus*
UTC	*Universal Time Coordinate*
UICC	*Universal Integrated Circuit Card*
W3C	*World Wide Web Consortium*
WSDL	*Web Services Description Language*
WWW	*World Wide Web*
XML	*Extensible Markup Language*



Referencias rápidas: Utiliza los anexos para localizar rápidamente esa palabra clave o esa clase que no recuerdas.



Trivial programación Android: Instálate esta app y mide tus conocimientos jugando en red contra otros oponentes.

De forma adicional, en la web www.androidcurso.com encontrarás:

- **Tutoriales sobre Java:** ¿Sabes lo que es la herencia, el polimorfismo o la sobrecarga en Java? Si no dominas el lenguaje de programación Java, te recomendamos que realices alguno de los tutoriales propuestos.
- **Código abierto de proyectos Android:** Muchos alumnos que han realizado un curso basado en este libro han tenido la generosidad de compartir sus proyectos con todos nosotros. Te recomendamos que consultes la lista de proyectos disponibles de código abierto: puedes aprender mucho estudiando su código. Cuando termines de leer este libro, también tú podrás hacer un proyecto como los que se muestran.
- **Material adicional sobre Android:** Encontrarás, además, nuevos tutoriales, vídeos, referencias, etc., no incluidos en el libro.
- **Cursos *online*:** Si te interesa ampliar tu formación, puedes matricularte en cursos sobre Android impartidos por la Universidad Politécnica de Valencia en la plataforma EdX. Incluso puedes obtener un título de Especialización o de Máster de forma 100 % *online*.

CAPÍTULO 1.

Visión general y entorno de desarrollo

La telefonía móvil está cambiando la sociedad actual de una forma tan significativa como lo ha hecho Internet. Esta revolución no ha hecho más que empezar; los nuevos terminales ofrecen unas capacidades similares a un ordenador personal, lo que permite que puedan ser utilizados para leer el correo o navegar por Internet. Pero, a diferencia de un ordenador, un teléfono móvil siempre está en el bolsillo del usuario. Esto permite un nuevo abanico de aplicaciones mucho más cercanas al usuario. De hecho, muchos autores coinciden en afirmar que el nuevo ordenador personal del siglo XXI será un terminal móvil.

El lanzamiento de Android como nueva plataforma para el desarrollo de aplicaciones móviles ha causado una gran expectación y ha tenido una importante aceptación tanto por parte de los usuarios como por parte de la industria. En la actualidad se ha convertido en la alternativa dominante frente a otras plataformas como iPhone o Windows Phone.

A lo largo de este capítulo veremos las características de Android que lo hacen diferente de sus competidores. Se explicará también cómo instalar y trabajar con el entorno de desarrollo (Android Studio).



Objetivos:

- Conocer las características de Android, destacando los aspectos que lo hacen diferente de sus competidores.
- Estudiar la arquitectura interna de Android.

- Aprender a instalar y trabajar con el entorno de desarrollo (Android SDK).
- Enumerar las principales versiones de Android y aprender a elegir la más idónea para desarrollar nuestras aplicaciones.
- Crear una primera aplicación y estudiar su estructura de un proyecto en Android.
- Conocer dónde podemos conseguir documentación sobre Android.
- Aprender a utilizar herramientas para detectar errores en el código.

1.1. ¿Qué hace que Android sea especial?

Como hemos comentado, existen muchas plataformas para móviles (Apple iOS, Windows Phone, BlackBerry, Palm, Java Micro Edition, Linux Mobile (LiMo), Firefox OS, etc.); sin embargo, Android presenta una serie de características que lo hacen diferente. Es el primero que combina en una misma solución las siguientes cualidades:

- **Plataforma abierta.** Es una plataforma de desarrollo libre basada en Linux y de código abierto. Una de sus grandes ventajas es que se puede usar y customizar el sistema sin pagar *royalties*.
- **Adaptable a diversos tipos de *hardware*.** Android no ha sido diseñado exclusivamente para su uso en teléfonos y tabletas. Hoy en día podemos encontrar relojes, gafas, cámaras, TV, sistema para automóviles, electrodomésticos y una gran variedad de sistemas empotrados que se basan en este sistema operativo, lo cual tiene sus evidentes ventajas, pero también va a suponer un esfuerzo adicional para el programador. La aplicación ha de funcionar correctamente en dispositivos con una gran variedad de tipos de entrada, pantalla, memoria, etc. Esta característica contrasta con la estrategia de Apple: en iOS tenemos que desarrollar una aplicación para iPhone y otra diferente para iPad.
- **Portabilidad asegurada.** Las aplicaciones finales son desarrolladas en Java, lo que nos asegura que podrán ser ejecutadas en cualquier tipo de CPU, tanto presente como futuro. Esto se consigue gracias al concepto de máquina virtual.
- **Arquitectura basada en componentes inspirados en Internet.** Por ejemplo, el diseño de la interfaz de usuario se hace en XML, lo que permite que una misma aplicación se ejecute en un reloj de pantalla reducida o en un televisor.
- **Filosofía de dispositivo siempre conectado a Internet.** Muchas aplicaciones solo funcionan si disponemos de una conexión permanente a Internet. Por ejemplo, comunicaciones interpersonales o navegación con mapas.

- **Gran cantidad de servicios incorporados.** Por ejemplo, localización basada tanto en GPS como en redes, bases de datos con SQL, reconocimiento y síntesis de voz, navegador, multimedia, etc.
- **Aceptable nivel de seguridad.** Los programas se encuentran aislados unos de otros gracias al concepto de ejecución dentro de una caja, que hereda de Linux. Además, cada aplicación dispone de una serie de permisos que limitan su rango de actuación (servicios de localización, acceso a Internet, etc.). Desde la versión 6.0 el usuario puede conceder o retirar permisos a las aplicaciones en cualquier momento.
- **Optimizado para baja potencia y poca memoria.** En el diseño de Android se ha tenido en cuenta el *hardware* específico de los dispositivos móviles. Por ejemplo, Android utiliza la máquina virtual ART (o Dalvik en versiones antiguas). Se trata de una implementación de Google de la máquina virtual Java optimizada para dispositivos móviles.
- **Alta calidad de gráficos y sonido.** Gráficos vectoriales suavizados, animaciones, gráficos en 3D basados en OpenGL. Incorpora los códecs estándares más comunes de audio y vídeo, incluyendo H.264 (AVC), MP3, AAC, etc.

Como hemos visto, Android combina características muy interesantes. No obstante, la pregunta del millón es: ¿se convertirá Android en el sistema operativo (SO) estándar para dispositivos móviles? Para contestar a esta pregunta habrá que ver la evolución del iPhone de Apple y cuál es la respuesta de Windows con el lanzamiento de su SO para móviles. No obstante, Android ha alcanzado un 85 % de cuota de mercado (90 % en España), cosa que lo deja en una posición predominante que es difícil que pierda a corto plazo.

En conclusión, Android nos ofrece una forma sencilla y novedosa de implementar potentes aplicaciones para diferentes tipos de dispositivos. A lo largo de este texto trataremos de mostrar de la forma más sencilla posible cómo conseguirlo.

1.2. Los orígenes

Google adquiere Android Inc. en el año 2005. Se trataba de una pequeña compañía, recién creada, orientada a la producción de aplicaciones para terminales móviles. Ese mismo año empiezan a trabajar en la creación de una máquina virtual Java optimizada para móviles (Dalvik VM).

En el año 2007 se crea el consorcio Open Handset Alliance[1] con el objetivo de desarrollar estándares abiertos para móviles. Está formado por Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericsson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel y otros. Uno de los objetivos fundamentales de esta alianza es promover el diseño y la difusión de la plataforma Android. Sus miembros se han

[1] http://www.openhandsetalliance.com

comprometido a publicar una parte importante de su propiedad intelectual como código abierto bajo licencia Apache v2.0.

En noviembre de 2007 se lanza una primera versión del Android SDK. Al año siguiente aparece el primer móvil con Android (T-Mobile G1). En octubre, Google libera el código fuente de Android, principalmente bajo licencia de código abierto Apache (licencia GPL v2 para el núcleo). Ese mismo mes se abre Android Market, para la descarga de aplicaciones. En abril de 2009, Google lanza la versión 1.5 del SDK, que incorpora nuevas características como el teclado en pantalla. A finales de 2009 se lanza la versión 2.0 y a lo largo de 2010, las versiones 2.1, 2.2 y 2.3.

Durante el año 2010, Android se consolida como uno de los sistemas operativos para móviles más utilizados, con resultados cercanos a iOS e incluso superando al sistema de Apple en EE.UU.

En el año 2011 se lanza la versión 3.x (Honeycomb), específica para tabletas, y la 4.0 (Ice Cream Sandwich), tanto para móviles como para tabletas. Durante ese año, Android se consolida como la plataforma para móviles más importante y alcanza una cuota de mercado superior al 50 %.

En 2012, Google cambia su estrategia en su tienda de descargas *online*, reemplazando Android Market por Google Play Store, donde en un solo portal unifica tanto la descarga de aplicaciones como la de contenidos. Ese año aparecen las versiones 4.1 y 4.2 (Jelly Bean). Android mantiene su espectacular crecimiento y alcanza, a finales de año, una cuota de mercado del 70 %.

En 2013 se lanzan las versiones 4.3 y 4.4 (KitKat). En 2014 se lanza la versión 5.0 (Lollipop). A finales de ese año, la cuota de mercado de Android llega al 85 %. En octubre de 2015 ha aparecido la versión 6.0, con el nombre de Marshmallow. En 2016 se lanzó la versión 7.0, Android Nougat. A finales de 2017 aparece la versión 8.0, con nombre Android Oreo. En agosto de 2018 se lanza la versión 9.0, Android Pie. En 2019 se lanza Android 10 y se abandonan así los nombres de dulces.



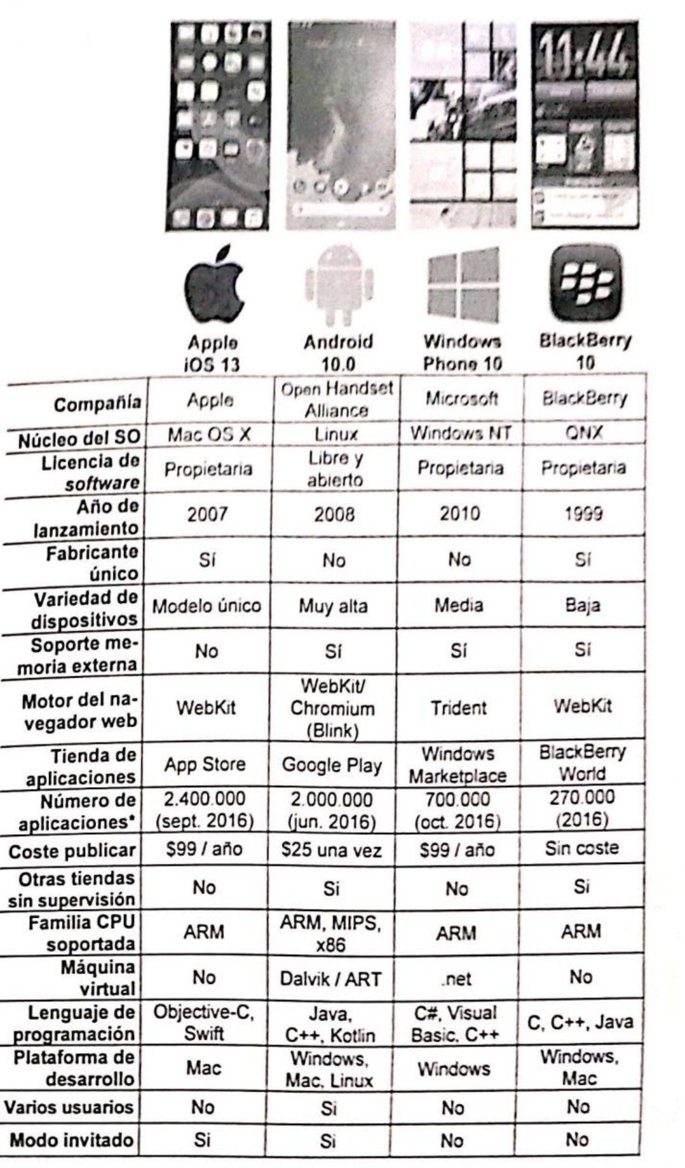
Vídeo[tutorial]: *Introducción a la plataforma Android*



Preguntas de repaso: *Características y orígenes de Android*

1.3. Comparativa con otras plataformas

En este apartado vamos a describir las características de las principales plataformas móviles disponibles en la actualidad. Dado la gran cantidad de datos que se indican, hemos utilizado una tabla para representar la información. De esta forma resulta más sencillo comparar las plataformas.



	Apple iOS 13	Android 10.0	Windows Phone 10	BlackBerry 10
Compañía	Apple	Open Handset Alliance	Microsoft	BlackBerry
Núcleo del SO	Mac OS X	Linux	Windows NT	QNX
Licencia de *software*	Propietaria	Libre y abierto	Propietaria	Propietaria
Año de lanzamiento	2007	2008	2010	1999
Fabricante único	Sí	No	No	Sí
Variedad de dispositivos	Modelo único	Muy alta	Media	Baja
Soporte memoria externa	No	Sí	Sí	Sí
Motor del navegador web	WebKit	WebKit/ Chromium (Blink)	Trident	WebKit
Tienda de aplicaciones	App Store	Google Play	Windows Marketplace	BlackBerry World
Número de aplicaciones*	2.400.000 (sept. 2016)	2.000.000 (jun. 2016)	700.000 (oct. 2016)	270.000 (2016)
Coste publicar	$99 / año	$25 una vez	$99 / año	Sin coste
Otras tiendas sin supervisión	No	Sí	No	Sí
Familia CPU soportada	ARM	ARM, MIPS, x86	ARM	ARM
Máquina virtual	No	Dalvik / ART	.net	No
Lenguaje de programación	Objective-C, Swift	Java, C++, Kotlin	C#, Visual Basic, C++	C, C++, Java
Plataforma de desarrollo	Mac	Windows, Mac, Linux	Windows	Windows, Mac
Varios usuarios	No	Sí	No	No
Modo invitado	Sí	Sí	No	No

Tabla 1: *Comparativa de las principales plataformas móviles (*Fuente www.statista.com).*

Otro aspecto fundamental a la hora de comparar las plataformas móviles es su cuota de mercado. En la siguiente gráfica podemos ver un estudio realizado por la empresa Gartner Group, donde se muestra la evolución del mercado de los sistemas operativos para móviles según el número de terminales vendidos. Podemos destacar la desaparición de la plataforma Symbian de Nokia, el declive continuo de BlackBerry, el estancamiento de la plataforma de Windows, que parece que no despega, y el afianzamiento de la cuota de mercado de Apple en torno al 15 %. En la gráfica se puede apreciar como Apple consigue anualmente un aumento significativo de ventas coincidiendo con el lanzamiento de un nuevo terminal. Finalmente, cabe señalar el espectacular ascenso de la plataforma Android, que en seis años ha alcanzado una cuota de mercado superior al 80 %.

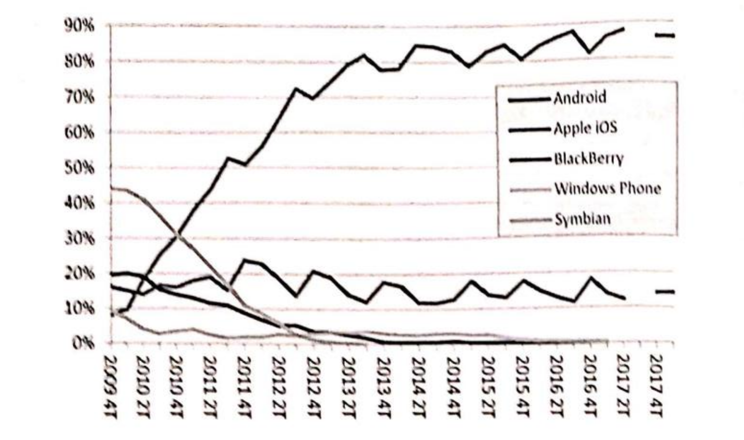


Figura 1: *Porcentaje de teléfonos inteligentes vendidos en todo el mundo, hasta el primer trimestre de 2018, según su sistema operativo (fuente: Gartner Group).*



Video[tutorial]: *Comparativa de las plataformas para móviles*


Preguntas de repaso: *Plataformas para móviles*

1.4. Arquitectura de Android

El siguiente gráfico muestra la arquitectura de Android. Como se puede ver, está formada por cuatro capas. Una de las características más importantes es que todas las capas están basadas en software libre.

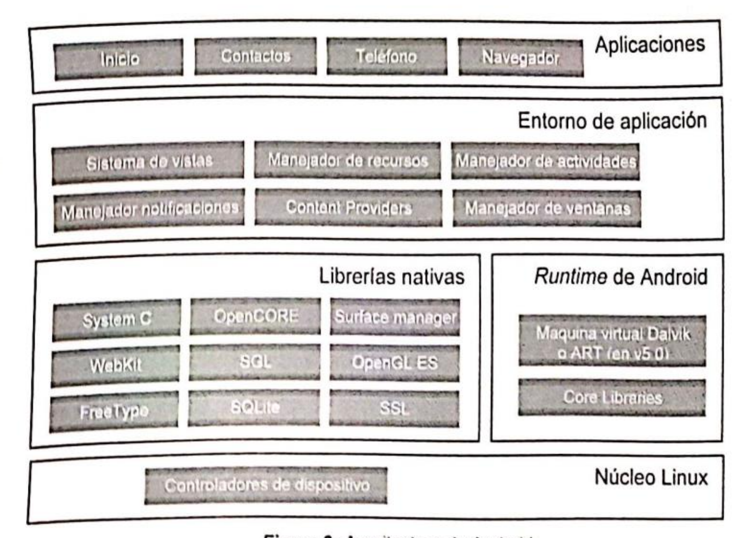


Figura 2: *Arquitectura de Android.*

1.4.1. El núcleo Linux

El núcleo de Android está formado por el sistema operativo Linux, versión 2.6. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de *drivers* para dispositivos.

Esta capa del modelo actúa como capa de abstracción entre el *hardware* y el resto de la pila. Por lo tanto, es la única dependiente del *hardware*.

1.4.2. *Runtime* de Android

Está basado en el concepto de máquina virtual utilizado en Java. Dadas las limitaciones de los dispositivos donde ha de correr Android (poca memoria y procesador limitado), no fue posible utilizar una máquina virtual Java estándar. Google tomó la decisión de crear una nueva, la máquina virtual Dalvik, que respondiera mejor a estas limitaciones.

Entre las características de la máquina virtual Dalvik que facilitan esta optimización de recursos se encuentra la ejecución de ficheros Dalvik ejecutables (*.dex*) –formato optimizado para ahorrar memoria–. Además, está basada en registros. Cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina virtual Dalvik. Delega al kernel de Linux algunas funciones como *threading* y el manejo de la memoria a bajo nivel.

A partir de Android 5.0 se reemplaza Dalvik por ART. Esta nueva máquina virtual consigue reducir el tiempo de ejecución del código Java hasta en un 33 %.

También se incluye en el *runtime* de Android el módulo *Core Libraries*, con la mayoría de las librerías disponibles en el lenguaje Java.

1.4.3. Librerías nativas

Incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android. Están compiladas en código nativo del procesador. Muchas de las librerías utilizan proyectos de código abierto. Algunas de estas librerías son:

- **System C library**: una derivación de la librería BSD de C estándar (libc), adaptada para dispositivos embebidos basados en Linux.
- **Media Framework**: librería basada en OpenCORE de PacketVideo. Soporta códecs de reproducción y grabación de multitud de formatos de audio y vídeo e imágenes MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.
- **Surface Manager**: maneja el acceso al subsistema de representación gráfica en 2D y 3D.
- **WebKit/Chromium**: soporta el navegador web utilizado en Android y en la vista *WebView*. En la versión 4.4, WebKit ha sido reemplazada por Chromium/Blink, que es la base del navegador Chrome de Google.
- **SGL**: motor de gráficos 2D.
- **Librerías 3D**: implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador *hardware* 3D si está disponible, o el *software* altamente optimizado de proyección 3D.
- **FreeType**: fuentes en *bitmap* y renderizado vectorial.
- **SQLite**: potente y ligero motor de bases de datos relacionales disponible para todas las aplicaciones.
- **SSL**: proporciona servicios de encriptación *Secure Socket Layer* (capa de conexión segura).

1.4.4. Entorno de aplicación

Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones, etc.).

Esta capa ha sido diseñada para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a las restricciones de seguridad). Este mismo mecanismo permite a los usuarios reemplazar componentes.

Los servicios más importantes que incluye son:

- **Views**: extenso conjunto de vistas (parte visual de los componentes).
- **Resource Manager**: proporciona acceso a recursos que no son en código.
- **Activity Manager**: maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.

- **Notification Manager**: permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
- **Content Providers**: mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).

Una de las mayores fortalezas del entorno de aplicación de Android es que se aprovecha el lenguaje de programación Java. El SDK de Android no acaba de ofrecer para su estándar todo lo disponible del entorno de ejecución Java (JRE), pero es compatible con una fracción muy significativa de este.

1.4.5. Aplicaciones

Este nivel está formado por el conjunto de aplicaciones instaladas en una máquina Android. Todas las aplicaciones han de correr en la máquina virtual ART para garantizar la seguridad del sistema.

Normalmente las aplicaciones Android están escritas en Java o Kotlin. Para desarrollar este tipo de aplicaciones podemos utilizar el Android SDK. Existe otra opción consistente en desarrollar las aplicaciones utilizando C/C++. Para esta opción podemos utilizar el Android NDK (*Native Development Kit*)[2].



Vídeo[tutorial]: *La arquitectura de Android*


Preguntas de repaso: *La arquitectura de Android*

1.5. Instalación del entorno de desarrollo

Google ha preparado el paquete de *software* **Android SDK**, que incorpora todas las herramientas necesarias para el desarrollo de aplicaciones en Android. En él se incluye: conversor de código, depurador, librerías, emuladores, documentación, etc. Todas estas herramientas son accesibles desde la línea de comandos.

No obstante, la mayoría de los desarrolladores prefieren utilizar un IDE (entorno de desarrollo integrado). Un IDE agrupa, en un entorno visual, un editor de código con todas las herramientas de desarrollo. Google recomienda utilizar Android Studio (basado en el IDE IntelliJ IDEA).

1.5.1. Instalación de la máquina virtual Java

Las aplicaciones Android están basadas en Java, por lo que necesitas instalar un *software* para ejecutar código Java en tu equipo. Este *software* se conoce como

[2] Para más información consultar *el Gran Libro de Android Avanzado*

máquina virtual Java, entorno de ejecución Java, Java Runtime Environment (JRE) o Java Virtual Machine (JVM).

Es muy posible que ya tengas instalada la máquina virtual Java en tu equipo. Si es así, puedes pasar directamente a uno de los apartados siguientes. En caso de dudas, puedes pasar también al punto siguiente. Al concluirlo te indicará si la versión de la máquina virtual Java es incorrecta. En caso necesario, regresa a este punto para instalar una que sea adecuada.

Para instalar la máquina virtual Java accede a http://www.java.com/es/download/, descarga e instala el fichero correspondiente a tu sistema operativo.

1.5.2. Instalación de Android Studio

En la edición de Google I/O 2014 se lanzó la primera versión estable de Android Studio. Se trata de un entorno de desarrollo para Android basado en el IDE IntelliJ IDEA. Entre las novedades introducidas destacamos:

- Construcción de proyectos usando la herramienta Gradle.
- Previsualización simultánea de un layout en varios tipos de dispositivos.
- Facilidades para el testeo de código basado en JUnit.
- Integración con herramientas de gestión de versiones (como GitHub).
- Desarrollo en un mismo proyecto de diferentes versiones (como Android Wear, Android TV y Android Auto).



Ejercicio: *Instalación de Android Studio*

NOTA: *Puedes encontrar una descripción más detallada de la instalación en https://developer.android.com/studio/install.html*

1. Descarga el paquete correspondiente a tu versión de la siguiente dirección:

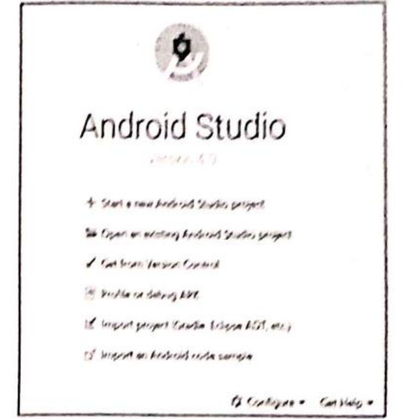
   http://developer.android.com/sdk/

2. Ejecuta el fichero obtenido en el paso anterior.
3. Selecciona todos los componentes a instalar y pulsa *Next*.

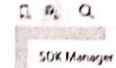
   Select components to install: Android Studio / ☑ Android SDK / ☑ Android Virtual Device

4. Acepta el contrato de licencia y selecciona las carpetas donde quieres instalar el IDE Android Studio y el SDK. En el resto de ventanas puedes utilizar las opciones por defecto. En la última ventana indica que quieres arrancar Android Studio.
5. Primero te preguntará si quieres importar la configuración desde una instalación anterior. Luego verificará si hay actualizaciones del SDK.

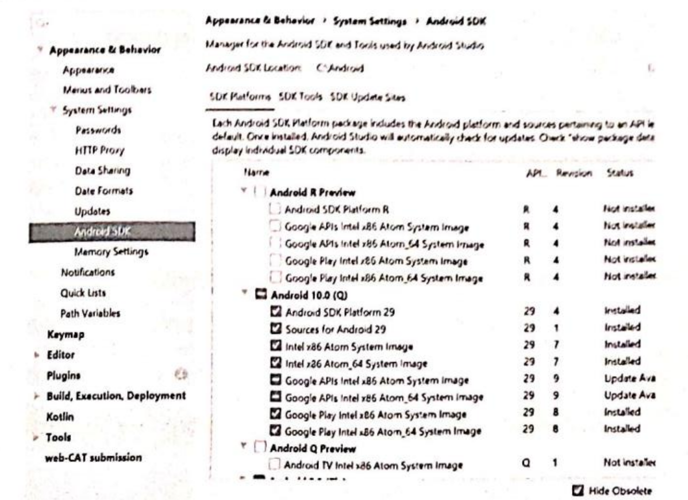
6. Tras pulsar en *Finish* pasamos a la ventana de bienvenida:



7. Comienza pulsando en *Configure*. Aparecerán varias opciones, selecciona *SDK Manager*. Esta herramienta es de gran utilidad para verificar si existen actualizaciones del SDK o nuevas versiones de la plataforma. Podrás acceder a ella desde la ventana principal de Android Studio pulsando en el botón *SDK Manager*.



8. Al entrar en el SDK Manager te muestra los paquetes instalados y los que puedes instalar o actualizar:



En la lengüeta *SDK Plataforms* se muestran los paquetes de plataforma. Pulsa en *Show Package Details* para ver los diferentes paquetes. Siempre es conveniente que tengas instalados los siguientes paquetes de la última plataforma disponible:

- *Android SDK Platform X (donde X es la última versión disponible)*
- *Sources for Android X (no es imprescindible)*
- *Google APIs … System Image (para crear emuladores con Google APIs)*
- *Google Play … System Image (para crear emuladores con Google APIs + Google Play)*

En la lengüeta *SDK Tools* se muestran paquetes con herramientas de la plataforma. Siempre es conveniente que tengas actualizados los siguientes paquetes:

- *Android SDK Build-Tools*
- *Android SDK Platform-tools*
- *Android SDK Tools*
- *Google Play services*



Recursos adicionales: *Teclas de acceso rápido en Android Studio*

Alt-Intro: Solución rápida (Ej. añade *imports* de las clases no resueltas).
Shift-F10 (**Ctrl-R** en Mac): Ejecuta el proyecto.
Shift-F9 (**Ctrl-D** en Mac): Depura el proyecto.
Shift-F6: Cambia el nombre de un identificador.
Ctrl-Alt-L (**Option-Cmd-L** en Mac): Formatea automáticamente el código.
Ctrl-Q (**F1** en Mac): Muestra documentación del código.
Ctrl-P: Muestra parámetros del método seleccionado.
F4 (**Cmd-↓** en Mac): Salta a declaración.
Ctrl-Y (**Cmd-Espacio** en Mac): Borra línea.
Alt-Insert (**Cmd-N** en Mac): Inserta método.



Enlaces de interés: *Conoce Android Studio*

https://developer.android.com/studio/intro/index.html?hl=es-419



Preguntas de repaso: *Instalación y entorno de desarrollo*

1.5.3. Creación de un dispositivo virtual Android (AVD)

Un dispositivo virtual Android (AVD) te va a permitir emular en tu ordenador diferentes tipos de dispositivos basados en Android. De esta forma podrás probar tus aplicaciones en una gran variedad de teléfonos, tabletas, relojes o TV con cualquier versión de Android, tamaño de pantalla o tipo de entrada.



Ejercicio: *Creación de un dispositivo virtual Android (AVD)*

1. Pulsa el botón *AVD Manager*.



Aparecerá la lista con los AVD creados. La primera vez estará vacía.

2. Pulsa a continuación el botón *Create Virtual Device…* para crear un nuevo AVD. Aparecerá la siguiente ventana:

Choose a device definition

Category	Name ▾	Play Store	Size	Resolution	Density
TV	Pixel XL		5.5"	1440x2...	560dpi
Phone	Pixel 3a XL		6.0"	1080x2...	400dpi
Wear OS	Pixel 3a	▶	5.6"	1080x2...	440dpi
Tablet	Pixel 3 XL		6.3"	1440x2...	560dpi
Autom...	Pixel 3	▶	5.46"	1080x2...	440dpi
	Pixel 2 XL		5.99"	1440x2...	560dpi

New Hardware Profile | Import Hardware Profiles

Pixel 2 — 1080px, 5.0", 1920px — Size: large, Ratio: long, Density: 420dpi

En la primera columna podremos seleccionar el tipo de dispositivo a emular (Google TV, móvil, dispositivo *wearable*, tableta o Auto). A la derecha, se muestran distintos dispositivos que emulan dispositivos reales de la familia Pixel y también otros genéricos. Junto al nombre de cada dispositivo, se indica si tiene la posibilidad de incorporar Google Play, el tamaño de la pantalla en pulgadas, la resolución y el tipo de densidad gráfica.

Si quieres añadir a esta lista un nuevo tipo de dispositivo, puedes seleccionar *New Hardware Profile*. Podrás indicar sus principales características y ponerle un nombre. Usando *Clone Device* podrás crear un nuevo tipo de AVD a partir del actual. Pulsando con el botón derecho sobre un tipo de dispositivo podrás eliminarlo o exportarlo a un fichero.

3. Pulsa *Next* para pasar a la siguiente ventana, donde podrás seleccionar la imagen del sistema que tendrá el dispositivo y el tipo de procesador:

Select a system image

Recommended | x86 Images | Other Images

Release Name	API Level	ABI	Target
R Download	R	x86	Android 10.0+ (Google Play)
Q	29	x86	Android 10.0 (Google Play)
Pie	28	x86	Android 9.0 (Google Play)
Oreo Download	27	x86	Android 8.1 (Google Play)
Oreo Download	26	x86	Android 8.0 (Google Play)
Nougat Download	25	x86	Android 7.1.1 (Google Play)
Nougat	24	x86	Android 7.0 (Google Play)



API Level
29
Android
10.0
Google Inc.
System Image
x86

Observa cómo las distintas versiones de Android se pueden seleccionar, solo con el código abierto de Android, añadiendo las API de Google (para utilizar servicios como Google Maps) o incluso incorporando Google Play (para poder instalar apps desde la tienda de Google).

4. Pulsa *Next* para pasar a la última ventana. Se nos mostrará un resumen con las opciones seleccionadas; además, podremos seleccionar la orientación inicial de AVD, si queremos usar el coprocesador gráfico (GPU) de nuestro ordenador o si queremos que dibuje un marco alrededor del emulador simulando un dispositivo real.

5. Pulsa en el botón *Show Avanced Settings* para que se muestren algunas configuraciones adicionales:

Camera — Front: Emulated; Back: VirtualScene

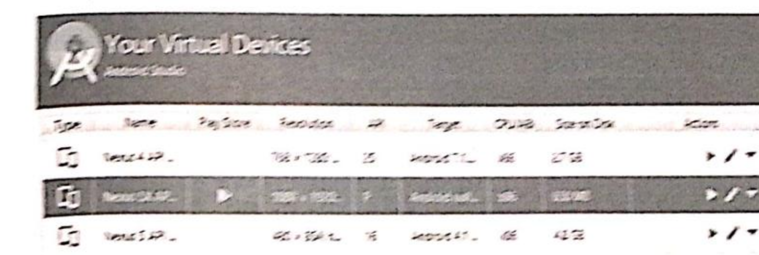
Network — Speed: Full; Latency: None

Emulated Performance — Graphics; Boot option: Cold boot / Quick boot

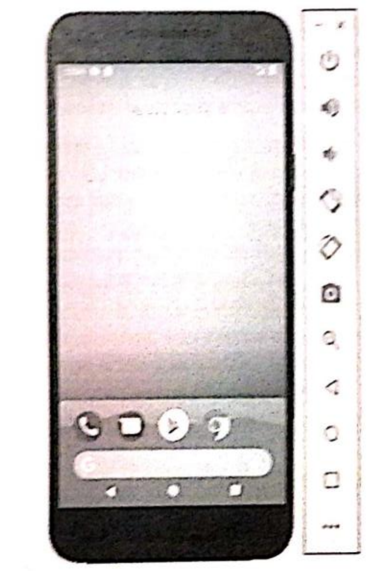
Memory and Storage — RAM: 1536; VM heap: 256; Internal Storage: 2048 MB; SD card: 512 MB

Podemos hacer que el emulador utilice la cámara o teclado de nuestro ordenador. También podemos imitar la velocidad y latencia en el acceso a la red. Finalmente, podremos ajustar la memoria utilizada: RAM total del dispositivo, memoria dinámica usada por Java y memoria para almacenamiento, tanto interna como externa.

6. Una vez introducida la configuración deseada, pulsa el botón Finish. Aparecerá el dispositivo creado en la lista:



7. Para arrancarlo, pulsa el botón con forma de triángulo verde que encontrarás en la columna de la derecha. Es posible que te pregunte por la entrada de vídeo para emular la cámara del AVD.



Vídeo[tutorial]: *Creación de dispositivos virtuales (AVD)*



Recursos adicionales: *Teclas de acceso rápido en un emulador*

Inicio: Tecla *Home*.

F2: Tecla *Menú*.

Esc: Tecla de volver.

F7: Tecla *On/Off*

Ctrl-F5/Ctrl-F6 o **KeyPad +/-:** Control de volumen de audio.

Ctrl-F11 o **KeyPad 7:** Cambia la orientación entre horizontal y vertical.

1.6. Las versiones de Android y niveles de API

Antes de empezar a programar en Android hay que elegir la versión del sistema para la que deseamos realizar la aplicación. Es muy importante observar que hay clases y métodos que están disponibles a partir de una versión; si las vamos a usar, hemos de conocer la versión mínima necesaria.

Cuando se ha lanzado una nueva plataforma, siempre ha sido compatible con las versiones anteriores. Es decir, solo se añaden nuevas funcionalidades, y en el caso de modificar alguna funcionalidad, no se elimina, sino que se etiqueta como obsoleta, pero normalmente se puede continuar utilizando.

A continuación, se describen las plataformas lanzadas hasta la fecha, con una breve descripción de las novedades introducidas. Las plataformas se identifican de tres formas alternativas: versión, nivel de API y nombre comercial. El nivel de API corresponde a números enteros, comenzando desde 1. Para los nombres comerciales se han elegido postres en orden alfabético: Cupcake (v1.5), Donut (v1.6), Éclair (v2.0), Froyo (v2.2), Gingerbread (v2.3), etc. Las dos primeras versiones, que hubieran correspondido a las letras A y B, no recibieron nombre.



Vídeo[tutorial]: *Descripción de las versiones de Android*

1.6.1. Las primeras versiones

Android 1.0 Nivel de API 1 (septiembre 2008)

Primera versión de Android. Nunca se utilizó comercialmente, por lo que no tiene mucho sentido desarrollarla para esta plataforma.

Android 1.1 Nivel de API 2 (febrero 2009)

No se añadieron apenas funcionalidades: simplemente se arreglaron algunos errores de la versión anterior. Es la opción a escoger si queremos desarrollar una aplicación compatible con todos los dispositivos Android. No obstante, apenas existen usuarios con esta versión.

1.6.2. Cupcake



Android 1.5 Nivel de API 3 (abril 2009)

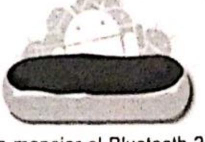
Es la primera versión con algún usuario, aunque en la actualidad apenas quedan. Como novedades, se incorpora la posibilidad de teclado en pantalla con predicción de texto (ya no es necesario que los terminales tengan un teclado físico), así como la capacidad de grabación avanzada de audio y vídeo. También aparecen los *widgets* de escritorio y *live folders*. Incorpora soporte para Bluetooth estéreo, por lo que permite conectarse automáticamente a auriculares Bluetooth. Las transiciones entre ventanas se realizan mediante animaciones.

1.6.3. Donut

Android 1.6 Nivel de API 4 (septiembre 2009)

Permite capacidades de búsqueda avanzada en todo el dispositivo. También se incorpora *gestures* y la síntesis de texto a voz. Asimismo, se facilita que una aplicación pueda trabajar con diferentes densidades de pantalla. Soporte para resolución de pantallas WVGA. Aparece un nuevo atributo XML, `onClick`, que puede especificarse en una vista. Soporte para CDMA/EVDO, 802.1x y VPNs.

1.6.4. Éclair



Android 2.0 Nivel de API 5 (octubre 2009)

Esta versión de API apenas cuenta con usuarios, dado que la mayoría de los fabricantes pasaron directamente de la versión 1.6 a la 2.1. Como novedades cabría destacar que incorpora una API para manejar el Bluetooth 2.1. Ofrece un servicio centralizado de manejo de cuentas. Se aumenta el número de tamaños de ventana y resoluciones soportadas. Nueva interfaz del navegador y soporte para HTML5. Mejoras en el calendario y soporte para Microsoft Exchange. La clase `MotionEvent` ahora soporta eventos en pantallas multitáctil.

Android 2.1 Nivel de API 7 (enero 2010)

Se considera una actualización menor, por lo que la siguieron llamando Éclair. Destacamos el reconocimiento de voz, que permite introducir un campo de texto dictando sin necesidad de utilizar el teclado. También permite desarrollar fondos de pantalla animados. Se puede obtener información sobre la señal de la red actual que posea el dispositivo. En el paquete WebKit se incluyen nuevos métodos para manipular bases de datos almacenadas en Internet.

1.6.5. Froyo

Android 2.2 Nivel de API 8 (mayo 2010)



Como característica más destacada se puede indicar la mejora de velocidad de ejecución de las aplicaciones (ejecución del código de la CPU de 2 a 5 veces más rápido que en la versión 2.1, de acuerdo con varios *benchmarks*). Esto se consigue con la introducción de un nuevo compilador JIT de la máquina Dalvik.

Se añaden varias mejoras relacionadas con el navegador web, como el soporte de Adobe Flash 10.1 y la incorporación del motor Javascript V8 utilizado en Chrome.

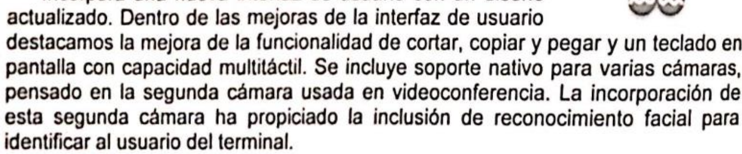
El desarrollo de aplicaciones permite las siguientes novedades: se puede preguntar al usuario si desea instalar una aplicación en un medio de almacenamiento externo (como una tarjeta SD), como alternativa a la instalación en la memoria interna del dispositivo; las aplicaciones se actualizan de forma automática cuando aparece una nueva versión; proporciona un servicio para la copia de seguridad de datos que se puede realizar desde la propia aplicación para garantizar al usuario el mantenimiento de sus datos; y por último, se facilita que las aplicaciones interaccionen con el reconocimiento de voz y que terceras partes proporcionen nuevos motores de reconocimiento.

Se mejora la conectividad: ahora podemos utilizar nuestro teléfono para dar acceso a Internet a otros dispositivos (*tethering*), tanto por USB como por Wi-Fi. También se añade el soporte a Wi-Fi IEEE 802.11n y notificaciones *push*.

Se añaden varias mejoras en diferentes componentes: en la API gráfica OpenGL ES; por ejemplo, se pasa a soportar la versión 2.0. Para finalizar, permite definir modos de interfaz de usuario («automóvil» y «noche») para que las aplicaciones se configuren según el modo seleccionado por el usuario.

1.6.6. Gingerbread

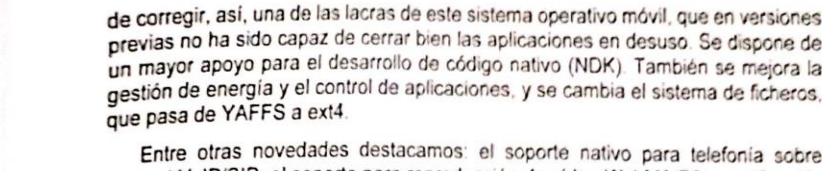
Android 2.3 Nivel de API 9 (diciembre 2010)



Debido al éxito de Android en las nuevas tabletas, ahora soporta mayores tamaños de pantalla y resoluciones (WXGA y superiores).

Incorpora una nueva interfaz de usuario con un diseño actualizado. Dentro de las mejoras de la interfaz de usuario destacamos la mejora de la funcionalidad de cortar, copiar y pegar y un teclado en pantalla con capacidad multitáctil. Se incluye soporte nativo para varias cámaras, pensado en la segunda cámara usada en videoconferencia. La incorporación de esta segunda cámara ha propiciado la inclusión de reconocimiento facial para identificar al usuario del terminal.

La máquina virtual Dalvik introduce un nuevo recolector de basura que minimiza las pausas de la aplicación, ayudando a garantizar una mejor animación y el aumento de la capacidad de respuesta en juegos y aplicaciones similares. Se trata



de corregir, así, una de las lacras de este sistema operativo móvil, que en versiones previas no ha sido capaz de cerrar bien las aplicaciones en desuso. Se dispone de un mayor apoyo para el desarrollo de código nativo (NDK). También se mejora la gestión de energía y el control de aplicaciones, y se cambia el sistema de ficheros, que pasa de YAFFS a ext4.

Entre otras novedades destacamos: el soporte nativo para telefonía sobre Internet VoIP/SIP; el soporte para reproducción de vídeo WebM/VP8 y codificación de audio AAC; el soporte para la tecnología NFC; las facilidades en el audio, los gráficos y las entradas para los desarrolladores de juegos; el soporte nativo para más sensores (como giroscopios y barómetros), y un gestor de descargas para las descargas largas.

1.6.7. Honeycomb

Android 3.0 Nivel de API 11 (febrero 2011)



Para mejorar la experiencia de Android en las nuevas tabletas se lanza la versión 3.0 optimizada para dispositivos con pantallas grandes. La nueva interfaz de usuario ha sido completamente rediseñada con paradigmas nuevos para la interacción y navegación. Entre las novedades introducidas destacan: los *fragments*, con los que podemos diseñar diferentes elementos de la interfaz de usuario; la barra de acciones, donde las aplicaciones pueden mostrar un menú siempre visible; las teclas físicas son reemplazadas por teclas en pantalla; se mejoran las notificaciones, arrastrar y soltar y las operaciones de cortar y pegar.

La nueva interfaz se pone a disposición de todas las aplicaciones, incluso las construidas para versiones anteriores de la plataforma. Esto se consigue gracias a la introducción de librerías de compatibilidad[3] que pueden ser utilizadas en versiones anteriores a la 3.0.

Se mejoran los gráficos 2D/3D gracias al renderizador OpenGL acelerado por *hardware*. Aparecerá el nuevo motor de gráficos Renderscript, que saca mayor rendimiento al *hardware* e incorpora su propia API. Se incorpora un nuevo motor de animaciones mucho más flexible, conocido como animación de propiedades.

Primera versión de la plataforma que soporta procesadores multinúcleo. La máquina virtual Dalvik ha sido optimizada para permitir multiprocesado, lo que permite una ejecución más rápida de las aplicaciones, incluso aquellas que son de hilo único.

Se incorporan varias mejoras multimedia, como listas de reproducción M3U a través de HTTP Live Streaming, soporte a la protección de derechos musicales (DRM) y soporte para la transferencia de archivos multimedia a través de USB con los protocolos MTP y PTP.

En esta versión se añaden nuevas alternativas de conectividad, como las nuevas API de Bluetooth A2DP para *streaming* de audio y HSP para conexiones

[3] http://developer.android.com/tools/support-library

seguras con dispositivos. También, se permite conectar teclados completos por USB o Bluetooth.

Se mejora el uso de los dispositivos en un entorno empresarial. Entre las novedades introducidas destacamos las nuevas políticas administrativas con encriptación del almacenamiento, caducidad de contraseña y mejoras para administrar los dispositivos de empresa de forma eficaz.

A pesar de la nueva interfaz gráfica optimizada para tabletas, Android 3.0 es compatible con las aplicaciones creadas para versiones anteriores.

Android 3.1 Nivel de API 12 (mayo 2011)

Se permite manejar dispositivos conectados por USB (tanto *host* como dispositivo). Protocolo de transferencia de fotos y vídeo (PTP/MTP) y de tiempo real (RTP).

Android 3.2 Nivel de API 13 (julio 2011)

Optimizaciones para distintos tipos de tableta. Zum compatible para aplicaciones de tamaño fijo. Sincronización multimedia desde SD.

1.6.8. Ice Cream Sandwich

Android 4.0 Nivel de API 14 (octubre 2011)



La característica más importante es que se unifican las dos versiones anteriores (2.x para teléfonos y 3.x para tabletas) en una sola compatible con cualquier tipo de dispositivo. A continuación destacamos algunas de las características más interesantes.

Se introduce una nueva interfaz de usuario totalmente renovada; por ejemplo, se reemplazan los botones físicos por botones en pantalla (como ocurría en las versiones 3.x). Nueva API de reconocimiento facial que, entre otras muchas aplicaciones, permite al propietario desbloquear el teléfono. También se mejora en el reconocimiento de voz; por ejemplo, se puede empezar a hablar sin esperar la conexión con el servidor.

Aparece un nuevo gestor de tráfico de datos por Internet, donde podremos ver el consumo de forma gráfica y donde podemos definir los límites de ese consumo para evitar cargos inesperados con la operadora. Incorpora herramientas para la edición de imágenes en tiempo real, para distorsionar, manipular e interactuar con la imagen en el momento de ser capturada. Se mejora la API para comunicaciones por NFC y la integración con redes sociales.

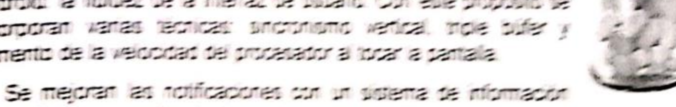
En diciembre de 2011 aparece una actualización de mantenimiento (versión 4.0.2) que no aumenta el nivel de API.

Android 4.0.3 Nivel de API 15 (diciembre 2011)

Se introducen ligeras mejoras en algunas API, incluyendo las de redes sociales, calendario, revisor ortográfico, texto a voz y bases de datos, entre otras. En marzo de 2012 aparece la actualización 4.0.4.

1.6.9. Jelly Bean

Android 4.1 Nivel de API 16 (julio 2012)



En esta versión se hace hincapié en mejorar un punto débil de Android: la fluidez de la interfaz de usuario. Con este propósito se incorporan varias técnicas: sincronismo vertical, triple búfer y aumento de la velocidad del procesador al tocar la pantalla.

Se mejoran las notificaciones con un sistema de información expandible personalizada. Los *widgets* de escritorio pueden ajustar su tamaño y hacerse sitio de forma automática al situarlos en el escritorio. El dictado por voz puede realizarse sin conexión a Internet (de momento, solo en inglés).

Se introducen varias mejoras en Google Search. Se potencia la búsqueda por voz con resultados en forma de ficha. La función Google Now permite utilizar información de posición, agenda y hora en las búsquedas.

Se incorporan nuevos soportes para usuarios internacionales, como texto bidireccional y teclados instalables. Para mejorar la seguridad, las aplicaciones son cifradas. También se permiten actualizaciones parciales de aplicaciones.

Android 4.2 Nivel de API 17 (noviembre 2012)

Una de las novedades más importantes es que podemos crear varias cuentas de usuario en el mismo dispositivo. Aunque esta característica solo está disponible en tabletas. Cada cuenta tendrá sus propias aplicaciones y su propia configuración.

Los *widgets* de escritorio pueden aparecer en la pantalla de bloqueo. Se incorpora un nuevo teclado predictivo deslizante al estilo Swype. Posibilidad de conectar dispositivo y TV HD mediante Wi-Fi (Miracast). Mejoras menores en las notificaciones. Nueva aplicación de cámara que incorpora la funcionalidad Photo Sphere para hacer fotos panorámicas inmersivas (en 360°).

Android 4.3 Nivel de API 18 (julio 2013)

Esta versión introduce mejoras en múltiples áreas. Entre ellas los *perfiles restringidos* (disponible solo en tabletas), que permiten controlar los derechos de los usuarios para ejecutar aplicaciones específicas y para tener acceso a datos específicos. Igualmente, los programadores pueden definir restricciones en las apps, que los propietarios pueden activar si quieren. Se da soporte para Bluetooth Low Energy (BLE), que permite a los dispositivos Android comunicarse con los periféricos con bajo consumo de energía. Se agregan nuevas características para la codificación, transmisión y multiplexación de datos multimedia. Se da soporte para OpenGL ES 3.0. Se mejora la seguridad para gestionar y ocultar las claves privadas y credenciales.

1.6.10. KitKat

Android 4.4 Nivel de API 19 (octubre 2013)



Aunque se esperaba la versión 5.0 y con el nombre de Key Lime Pie, Google sorprendió con el cambio de nombre, que se debió a un acuerdo con Nestlé para asociar ambas marcas.

El principal objetivo de la versión 4.4 es hacer que Android esté disponible en una gama aún más amplia de dispositivos, incluyendo aquellos con tamaños de memoria RAM de solo 512 MB. Para ello, todos los componentes principales de Android han sido recortados para reducir sus requerimientos de memoria, y se ha creado una nueva API que permite adaptar el comportamiento de la aplicación en dispositivos con poca memoria.

Más visibles son algunas nuevas características de la interfaz de usuario. El modo de inmersión en pantalla completa oculta todas las interfaces del sistema (barras de navegación y de estado), de tal manera que una aplicación pueda aprovechar el tamaño de la pantalla completa. *WebViews* (componente de la interfaz de usuario para mostrar las páginas web) se basa ahora en el *software* de Chrome de Google y, por lo tanto, puede mostrar contenido basado en HTML5.

Se mejora la conectividad con soporte de NFC para emular tarjetas de pago tipo HCE, varios protocolos sobre Bluetooth y soporte para mandos infrarrojos. También se mejoran los sensores para disminuir su consumo y se incorpora un sensor contador de pasos.

Se facilita el acceso de las aplicaciones a la nube con un nuevo marco de almacenamiento. Este marco incorpora un tipo específico de *content provider* conocido como *document provider*, nuevas intenciones para abrir y crear documentos y una ventana de diálogo que permite al usuario seleccionar ficheros. Se incorpora un administrador de impresión para enviar documentos a través de Wi-Fi a una impresora. También se añade un *content provider* para gestionar los SMS.

Desde una perspectiva técnica, hay que destacar la introducción de la nueva máquina virtual ART, que consigue tiempos de ejecución muy superiores a la máquina Dalvik. Sin embargo, todavía está en una etapa experimental. Por defecto se utiliza la máquina virtual Dalvik, y se permite a los programadores activar opcionalmente ART para verificar que sus aplicaciones funcionan correctamente.



Vídeo[tutorial]: *Android 4.4 KitKat*

1.6.11. Lollipop

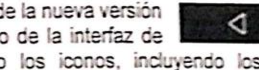
Android 5.0 Nivel de API 21 (noviembre 2014)

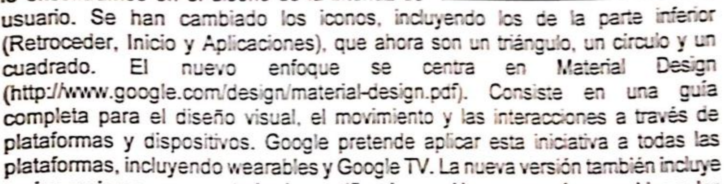


La novedad más importante de Lollipop es la extensión de Android a nuevas plataformas, incluyendo Android Wear, Android TV y Android Auto. Hay un cambio significativo en la arquitectura, al utilizar la máquina virtual ART en lugar de Dalvik. Esta novedad ya había sido incorporada en la versión anterior a modo de prueba. ART mejora de forma considerable el tiempo de ejecución del código escrito en Java. Además se soporta dispositivos de 64 bits en procesadores ARM, x86, y MIPS. Muchas aplicaciones del sistema (Chrome, Gmail, ...) se han incorporado en código nativo para una ejecución más rápida.

Desde el punto de vista del consumo de batería, hay que resaltar que en Lollipop el modo de ahorro de batería se activa por defecto. Este modo desconecta algunos componentes en caso de que la batería esté baja. Se incorpora una nueva API (*android.app.job.JobScheduler*) que nos permite que ciertos trabajos se realicen solo cuando se cumplan determinadas condiciones (por ejemplo con el dispositivo cargando). También se incluyen completas estadísticas para analizar el consumo que nuestras aplicaciones hacen de la batería.

En el campo Gráfico Android Lollipop incorpora soporte nativo para OpenGL ES 3.1. Además esta versión permite añadir a nuestras aplicaciones un paquete de extensión con funcionalidades gráficas avanzadas (fragment shader, tessellation, geometry shaders, ASTC, ...).





Otro aspecto innovador de la nueva versión lo encontramos en el diseño de la interfaz de usuario. Se han cambiado los iconos, incluyendo los de la parte inferior (Retroceder, Inicio y Aplicaciones), que ahora son un triángulo, un círculo y un cuadrado. El nuevo enfoque se centra en Material Design (http://www.google.com/design/material-design.pdf). Consiste en una guía completa para el diseño visual, el movimiento y las interacciones a través de plataformas y dispositivos. Google pretende aplicar esta iniciativa a todas las plataformas, incluyendo wearables y Google TV. La nueva versión también incluye varias mejoras para controlar las notificaciones. Ahora son más parecidas a las tarjetas de Google Now y pueden verse en la pantalla de bloqueo.

Se incorporan nuevos sensores como el de pulso cardiaco, el de inclinación (para reconocer el tipo de actividad del usuario), y sensores de interacción compuestos para detectar ciertos gestos.

Como curiosidad la nueva versión introduce un modo de bloqueo que impide al usuario salir de una aplicación y bloquea las notificaciones. Esto podría utilizarse, por ejemplo, para que mientras un usuario realiza un examen, no pueda ver las notificaciones, acceder a otras aplicaciones, o volver a la pantalla de inicio.



Vídeo[tutorial]: *Android 5.0 Lollipop*

Android 5.1 Nivel de API 22 (marzo 2015)

Se añaden algunas mejoras a nivel de usuario en los ajustes rápidos. A nivel de API se añade soporte para varias tarjetas SIM en un mismo teléfono; la clase AndroidHttpClient se marca como obsoleta; y se añade un API para que las empresas proveedoras de servicios de telecomunicación puedan distribuir *software* de forma segura a través de Google Play. La característica más interesante es que para poder acceder a esta API la aplicación ha de estar firmada con un certificado que coincida con el que el usuario tiene en su tarjeta UICC.

1.6.12. Marshmallow

Android 6.0 Nivel de API 23 (octubre 2015)

Una de las novedades más interesantes es el administrador de permisos. Los usuarios podrán conceder o retirar ciertos permisos a cada aplicación. Con esto el sistema da mucha más protección a la privacidad de los usuarios.

Ahora, el sistema realiza una copia de seguridad automática de todos los datos de las aplicaciones. Esto resulta muy útil al cambiar de dispositivo o tras restaurar valores de fábrica. Para disponer de esta funcionalidad simplemente usa el *target* Android 6.0. No es necesario agregar código adicional.

Android 6.0 integra el asistente por voz Now on Tap. Es una evolución de Google Now más integrada con las aplicaciones. Se activa con pulsación larga de home. Aparecerán tarjetas sobre la aplicación actual y lo que muestra. La aplicación actual podrá aportar información al asistente. En esta misma línea, se añade un API que permite interacciones basadas en voz. Es decir, si nuestra aplicación ha sido lanzada por voz, podremos solicitar una confirmación de voz del usuario, seleccionar de una lista de opciones o cualquier información que necesite.

Se introducen los enlaces de aplicación con los que podremos asociar la aplicación que abre una URL en función de su dominio web. Aunque muchos dispositivos ya lo permitían, en esta actualización se añade autentificación por huella digital a la API. Tu aplicación puede autentificar al usuario usando las credenciales para desbloquear su dispositivo (pin, patrón o contraseña). Esto libera al usuario de tener que recordar contraseñas específicas de la aplicación. Y te evita tener que implementar tu propia interfaz de autenticación.

Compartir con otros usuarios ahora es más fácil con Direct Share. Permite no solo escoger la aplicación con la que compartes, si no también el usuario. Si tu aplicación es un posible destino para compartir vas a poder indicar al sistema la lista de usuarios que pueden recibir información.

En Android 6.0 podemos utilizar parte de un dispositivo de almacenamiento externo, para que sea usado como almacenamiento interno. Podemos fragmentar, formatear y encriptar una tarjeta SD para ser usada como memoria interna. También podemos montar y extraer lápices de memoria USB de forma nativa.

Se incorpora la plataforma de pagos abierta *Android Pay* que combina NFC y Host Card Emulation. El nuevo gestor de batería, Doze, realiza un uso más eficiente de los recursos cuando el dispositivo está en reposo, con lo que podemos obtener dos horas extras de autonomía. Se da soporte de forma nativa a pantallas 4 K, lápices Bluetooth, múltiples tarjetas SIM y linterna. Mejoras de posicionamiento utilizando redes WiFi y dispositivos Bluetooth.



Vídeo[tutorial]: *Android 6.0 Marshmallow*

1.6.13. Android Nougat



Android 7.0 Nivel de API 24 (julio 2016)

Ahora los usuarios pueden abrir varias aplicaciones al mismo tiempo en la pantalla. Puedes configurar tu aplicación para que se visualice con unas dimensiones mínimas o inhabilitar la visualización de ventanas múltiples.

Las notificaciones han sido rediseñadas para un uso más ágil. Hay más opciones para personalizar el estilo de los mensajes (`MessageStyle`). Puedes agrupar notificaciones por temas o programar una respuesta directa.

En la versión anterior se utilizaba una estrategia de compilación *Ahead of Time* (AOT): cuando se descargaba una aplicación, su código era traducido de *bytecodes* a código nativo, lo que mejoraba los tiempos de ejecución. En la nueva versión se incorpora también la compilación *Just in Time* (JIT), donde no se compila hasta que el código va a ser ejecutado. Android 7.0 propone un planteamiento mixto según el perfil del código. Los métodos directos se compilan previamente (AOT), mientras que otras partes no se compilan hasta que se usan (JIT). Aunque AOT puede introducir retardos en ejecución, ahorra tiempo en la precompilación y en memoria. El mayor impacto de esta técnica se nota en la instalación de las aplicaciones y actualizaciones del sistema. Mientras que en Android 6.0 una actualización podría usar varios minutos, ahora se instala en cuestión de segundos.

Android Nougat incorpora la plataforma de realidad virtual *Daydream*. Se trata de una propuesta de Google que complementa la iniciativa *Cardboard*. Incluye especificaciones *software* y *hardware* que nos permitirán diferenciar a los dispositivos compatibles. Los principales fabricantes de móviles se han unido a esta iniciativa.

En la versión anterior, el gestor de batería Doze solo se activaba cuando el dispositivo estaba en reposo. Ahora, se activa poco tiempo después de apagarse la pantalla. Esto permite ahorrar batería cuando llevamos el dispositivo en el bolsillo.

También se ha añadido la nueva API para gráficos 3D, Vulcan, como alternativa a OpenGL. Minimiza la sobrecarga de CPU en el controlador, lo que permite aumentar la velocidad de los juegos.

El usuario va a poder activar el modo de ahorro de datos cuando se encuentre en itinerancia o cuando esté a punto de agotar un paquete de datos. En este caso, tanto el sistema como las aplicaciones han de tratar de minimizar al máximo las transferencias de datos.

Android 7.1 Nivel de API 25 (diciembre 2016)

La principal novedad son los accesos directos a aplicaciones. Desde el icono de la aplicación, con una pulsación prolongada, aparecen varias opciones que podremos seleccionar. Por ejemplo, podremos iniciar una navegación privada con Chrome de forma directa. Los accesos directos que quieras incorporar a tu aplicación, los podrás configurar por medio de *intens*, que deben especificarse en un fichero de configuración[4].

[4] https://developer.android.com/guide/topics/ui/shortcuts.html

Se incorporan otras novedades como la posibilidad de insertar imágenes desde el teclado, de la misma forma que ahora insertamos emoticonos.



Vídeo[tutorial]: *Android 7.0*

1.6.14. Android Oreo

Android 8.0 Nivel de API 26 (agosto 2017)



Destacan las siguientes mejoras en seguridad: se introduce Google Play Protect, que escanea regularmente las aplicaciones en busca de *malware*. La opción "Orígenes desconocidos" desaparece. Ahora podemos indicar qué aplicaciones pueden instalar apks y cuáles no. Desde la opción "Acceso especial de aplicaciones" podemos configurar qué aplicaciones pueden realizar ciertas acciones.

El sistema limita más los procesos en segundo plano para conseguir ahorro en la batería. Se mejora el tiempo de arranque del sistema.

Pensando en los países emergentes, se lanza Android Go: Una distribución adaptada para dispositivos de gama baja (1 GB de RAM o menos). Se preinstalan apps ligeras y en Google Play Store destacan aplicaciones ligeras adecuadas para estos dispositivos. Estas aplicaciones han de cubrir 3 requisitos: trabajar sin red, pesar menos de 10 MB y proporcionar un buen rendimiento de batería.

Con el fin de reducir la fragmentación de Android, aparece el proyecto Treble, que facilitará las actualizaciones a los fabricantes. Se reestructura la arquitectura de Android para definir una interfaz clara entre la capa del Núcleo Linux (con sus *drivers*) y las capas del Framework. Esto permite actualizar Android sin tener que tocar la capa del Núcleo Linux.

Las notificaciones presentan varias mejoras: Podemos añadir color de fondo. Se ordenan por importancia. Las aplicaciones pueden crear canales de notificaciones y el usuario decidir cuáles quiere recibir. Podemos posponer una notificación o verlas pulsando sobre el icono de la aplicación.

Los iconos tendrán que estar diseñados en dos capas: El icono y el fondo del icono. Esto permite adaptarse al dispositivo. Además, el usuario podrá escoger entre iconos circulares, cuadrados o de esquinas redondeadas.

Ahora podemos reproducir un vídeo en una ventana flotante mientras utilizamos otras aplicaciones. Al seleccionar un texto se nos sugieren acciones cuando se trata de un número de teléfono o una dirección. El Autocompletar de Google, que antes estaba disponible en Chrome para guardar contraseñas, ahora se puede usar en cualquier aplicación Android.

1.6.15. Android Pie

Android 9.0 Nivel de API 28 (agosto 2018)



Una de las novedades más interesantes es el nuevo API WiFi RTT introducido en IEEE 802.11mc. Permite estimar la distancia entre nuestro dispositivo y los puntos de acceso cercanos, lo que permite sistemas de posicionamiento en interiores con una precisión de 1 a 2 metros. Otro importante cambio es la navegación por gestos. Se reemplazan los tres botones en pantalla (triangulo, círculo y cuadrado) por solo 2 (retroceder e inicio). El botón de inicio admite diferentes gestos para ir al asistente de Google, cambiar entre apps recientes o abrir el menú de apps.

Una interesante innovación es el uso de Inteligencia Artificial, para mejorar diferentes aspectos. La idea consiste en aprender nuestros hábitos a la hora de usar las aplicaciones. Con esta información se puede quitar preferencia sobre el uso de la CPU a las apps menos utilizadas, consiguiendo una reducción de hasta un 30 %. Este menor uso de la CPU prolongará la vida de la batería. Usando técnicas similares se pretende aprender cuando el usuario va a arrancar una aplicación o una acción de esta. De esta forma el sistema puede cargar en memoria la aplicación antes incluso que el usuario decida utilizarla.

Se introducen algunas mejoras que fomentan un uso responsable y saludable del móvil. Por ejemplo, desde el Dashboard podemos consultar el uso que hacemos cada día, en cada aplicación. Podemos establecer alarmas de uso excesivo muy interesantes para el control parental. En esta línea, se introducen nuevos modos de relajación y no molestar para favorecer la desconexión digital.

1.6.16. Android 10

Android 10.0 Nivel de API 29 (septiembre 2019)



A partir de la versión 10, Google quiere simplificar la marca y los nombres de las versiones. Se abandonan los nombres de postre, para utilizar un simple número entero. Para el logo se usa solo la cabeza del robot, en un tono de verde algo más claro.

Ya no son necesarios los botones para la navegación a través del sistema operativo. Siguiendo la pauta propuesta en iOS, ahora se utiliza un control por gestos. Por ejemplo, para volver a la actividad anterior deslizaremos desde el extremo derecho a la izquierda.

Se introduce el Focus Mode que activaremos cuando queramos concentramos en una determinada tarea o juego y no queramos ser molestados. En este modo podemos configurar qué aplicaciones pueden lanzar una notificación y cuáles no. Implantación nativa del modo oscuro que permite un significativo ahorro de batería en pantallas OLED.

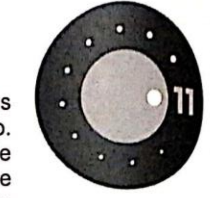
La función Live Caption permite que el sistema introduzca subtítulos de manera automática cuando se reproduce cualquier contenido de audio o vídeo. Está pensado

para reproducir estos contenidos cuando estamos en público y no queremos usar auriculares. Se obtienen de forma local por lo que no es necesario conexión a Internet. No obstante, solo puede activarse si disponemos de un procesador de gran potencia.

Para reducir la fragmentación, las actualizaciones de seguridad se instalan a través de Google Play, sin la intervención del fabricante. Aumentan las restricciones a las aplicaciones de fondo y se mejoran los permisos. Por ejemplo, podemos indicar que una aplicación tenga acceso a nuestra localización solo cuando esté en primer plano.

Se introduce soporte para 5G, WiFi 6, WPA3, teléfonos plegables y presión en pantallas táctiles.

1.6.17. Android 11



Android 11.0 Nivel de API 30 (septiembre 2020)

La última versión de Android incorpora interesantes novedades. Destacan varias mejoras en el interfaz de usuario. Con el nuevo menú de apagado (pulsación larga en botón de bloqueo) además de activar el modo avión o silenciar, se podrá acceder a tarjetas de pago, dispositivos domóticos y un largo etcétera. Las conversaciones en redes sociales se organizan en un apartado aparte de las notificaciones y pueden mostrarse en forma de burbuja flotante. Ahora se permite desde los controles grabar vídeos de la pantalla del dispositivo.

Se mejora la gestión de permisos. El usuario podrá dar el permiso para un solo uso y si lleva varios meses sin usar la aplicación tendrá que dar de nuevo los permisos. La funcionalidad Scoped Storage hace que cada aplicación acceda a un área de almacenamiento separada los que mejora la velocidad de lectura, la seguridad y evita tener que dar permiso a cada aplicación para almacenar datos.

La pila de Bluetooth ha sido rediseñada para evitar los frecuentes problemas de comunicación. Además, podemos seguir utilizándolo en modo avión.



Preguntas de repaso: *Las versiones de Android*

1.6.18. Elección de la plataforma de desarrollo



Vídeo[tutorial]: *Elegir la versión en una aplicación Android*

A la hora de seleccionar la plataforma de desarrollo hay que consultar si necesitamos alguna característica especial que solo esté disponible a partir de una versión. Todos los usuarios con versiones inferiores a la seleccionada no podrán instalar la aplicación. Por lo tanto, es recomendable seleccionar la menor versión posible que nuestra aplicación pueda soportar. Por ejemplo, si en nuestra aplicación queremos utilizar gráficos vectoriales, tendremos que utilizar la versión 5.0, al ser la primera que los soporta. El problema es que la aplicación no podrá ser instalada en dispositivos que tengan una versión anterior a la 5.0. Para ayudarnos a tomar la decisión de qué plataforma utilizar, puede ser interesante consultar los porcentajes de utilización de cada versión. Un gráfico similar al siguiente se muestra al crear una nueva aplicación:

ANDROID PLATFORM VERSION	API LEVEL	CUMULATIVE DISTRIBUTION
4.0	15	
4.1	16	99,8%
4.2	17	99,2%
4.3	18	98,4%
4.4 KitKat	19	98,1%
5.0	21	94,1%
5.1	22	92,3%
6.0	23	84,9%
7.0	24	73,7%
7.1	25	66,2%
8.0	26	60,8%
8.1	27	53,5%
9.0	28	39,5%
10 Android 10	29	8,2%

KitKat

Printing Framework
Print generic content
Print images
OEM print services

SMS Provider
Read and write SMS and MMS messages
Select default SMS app

Wireless and Connectivity
Host NFC card emulation
NFC reader mode
Infrared support

Multimedia
Adaptive video playback
On-demand audio timestamps
Surface image reader
Peak and RMS audio measurements
Loudness enhancer
Remote controllers
Closed captions

Animation and Graphics
Scenes and transitions
Animator pausing
Reusable bitmaps

User Content
Storage access framework
External storage access
Sync adapters

User Input
New sensor types including step
Batched sensor events
Controller identities

User Interface
Immersive full-screen mode
Translucent system bars
Enhanced notification listener
Live regions for accessibility

https://developer.android.com/about/versions/android-4.4.html

Figura 3: Porcentaje de dispositivos Android donde se podrá instalar una aplicación, según la versión mínima escogida. Los datos se obtienen a partir de los accesos realizados a Google Play. Versiones con un porcentaje inferior al 0,1 % no se contabilizan.

Tras estudiar la gráfica podemos extraer que prácticamente el 100 % de los dispositivos tienen una versión 4.0 o inferior. Con un sencillo cálculo puedes deducir que la versión 4.1 está instalada en un 0,2 % de dispositivos.

Como versión mínima para desarrollar nuestro proyecto podemos escoger la 4.0. Pero también es buena idea utilizar la 4.1, 4.2 o 4.4 para desarrollar nuestro proyecto, dado que daríamos cobertura de casi el 100 % de los terminales. La versión 9.0 es la mayoritaria con un 31,3 % de los dispositivos. No obstante, si la escogiéramos como versión mínima solo podrían instalarla el 39,5 % de los dispositivos. Estas cifras cambian mes a mes, por lo que recomendamos consultar el gráfico cada vez que vayamos a crear una nueva aplicación.



Enlaces de interés:

- ***Android Developers: Plataform Versions:*** Estadística de dispositivos Android, según la plataforma instalada, que han accedido a Android Market.

  http://developer.android.com/about/dashboards/index.html

- *Android Developers:* En el menú de la izquierda aparecen enlaces a las principales versiones de la plataforma. Si pulsas sobre ellos, encontrarás una descripción exhaustiva de cada plataforma.

  http://developer.android.com/about/index.html



Preguntas de repaso: *Elegir una versión de Android*

1.6.19. Las librerías de compatibilidad (*support library*)

Tal y como se ha descrito, la filosofía tradicional de Android ha sido que las novedades que aparecen en una API solo puedan usarse en dispositivos que soporten esa API. Como acabamos de ver, la fragmentación de las versiones de Android es muy grande, es decir, actualmente podemos encontrar dispositivos con una gran variedad de versiones. Con el fin de que la aplicación pueda ser usada por el mayor número posible de usuarios hemos de ser muy conservadores a la hora de escoger la versión mínima de API de nuestra aplicación. La consecuencia es que las novedades que aparecen en las últimas versiones de Android no pueden ser usadas.

En la versión 3.0 aparecieron importantes novedades que Google quería que se incorporaran en las aplicaciones lo antes posible (*fragments*, nuevas notificaciones, etc.). Con este fin creó las librerías de compatibilidad para poder incorporar ciertas funcionalidades en cualquier versión de Android.



Vídeo[tutorial]: *Las librerías de compatibilidad (support library)*

Desde la versión 9.0 las librerías de compatibilidad también se incluyen en las librerías AndroidX[5], que son parte del proyecto Jetpack[6]. En las librerías AndroidX se incluye tanto las librerías de compatibilidad como los componentes de Jetpack.

A diferencia de la librería de compatibilidad, cada paquete de AndroidX tiene su propia versión, y se mantienen y actualizan de manera separada. Todos los paquetes están en un espacio de nombre que empieza por `androidx.*`.

Algunos paquetes muy usados se muestran a continuación:

- **v4 Support Library:** (`androidx.legacy:legacy-support-v4`) Esta librería permitía utilizar muchas clases introducidas en la versión 3.0 cuando trabajábamos con un API mínimo. En la actualidad ya no es necesaria utilizarla, dado que ya es recomendable utilizar como API mínimo la versión 4.0 o, superior. Puede usarse en una aplicación con nivel de API 4 (v1.6) o superior. Incorpora las clases: `Fragment`, `NotificationCompat`,

[5] https://developer.android.com/jetpack/androidx?hl=ES

[6] https://developer.android.com/jetpack?hl=ES

  `LocalBroadcastManager`, `ViewPager`, `PagerTitleStrip`, `PagerTabStrip`, `DrawerLayout`, `SlidingPaneLayout`, `ExploreByTouchHelper`, `Loader` y `FileProvider`.

- **appcompat**: (`androidx.appcompat`) Permite utilizar un IU basado en la Barra de Acciones siguiendo especificaciones de Material Design. Se añade por defecto cuando creamos un nuevo proyecto. Incorpora las clases: `ActionBar`, `AppCompatActivity`, `AppCompatDialog` y `ShareActionProvider`.
- **recyclerview**: (`androidx.recyclerview`) Incorpora la vista `RecyclerView`, una versión mejorada que reemplaza a `ListView` y `GridView`.
- **constraintlayout**: Da soporte al layout `ConstraintLayout`.
- **preference**: (`androidx.preference`) Incorpora las clases `CheckBoxPreference` y `ListPreference` usadas en preferencias.
- **cardview**: (`androidx.cardview`) Incorpora la vista `CardView`, una forma estándar de mostrar información especialmente útil en Android Wear y TV.
- **palette**: (`androidx.palette`) Incorpora la clase `Palette`, que permite extraer los colores principales de una imagen.
- **mediarouter**: (`androidx.mediarouter`) Da soporte a Google Cast.
- **Design Support Library:** (`com.google.android.material`) Librería que incorpora varios componentes de Material Design.

Si tienes dudas sobre los nuevos paquetes utilizados consulta la siguiente tabla: https://developer.android.com/jetpack/androidx/migrate

1.7. Creación de un primer proyecto

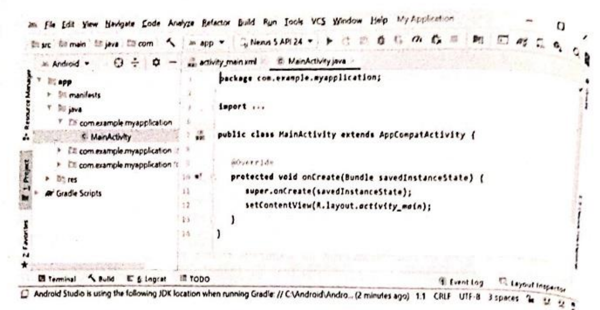
Utilizar un entorno de desarrollo nos facilita mucho la creación de aplicaciones. Esto es especialmente importante en Android dado que tendremos que utilizar una gran variedad de ficheros. Gracias a Android Studio, la creación y gestión de proyectos se realizará de forma muy rápida, acelerando los ciclos de desarrollo.



Ejercicio: *Crear un primer proyecto*

Para crear un primer proyecto Android, con Android Studio sigue los siguientes pasos:

1. Selecciona *File > New > New Project...*
2. En primer lugar, podrás indicar la plataforma para la que quieres desarrollar (teléfonos y tabletas, Wear OS, TV, ...) y el tipo de actividad inicial que quieres en tu aplicación:



Observa que la clase `MainActivity` extiende `AppCompactActivity` que a su vez es un descendiente de `Activity`.Una *actividad* es una entidad de aplicación que se utiliza para representar cada una de las pantallas de nuestra aplicación. Es decir, el usuario interactúa con solo una de estas actividades y va navegando entre ellas. El sistema llamará al método `onCreate()` cuando comience su ejecución. Es donde se debe realizar la inicialización y la configuración de la interfaz del usuario. Las actividades van a ser las encargadas de interactuar con el usuario.

Nota sobre Java/Kotlin: *Antes de este método se ha utilizado la anotación* `@Override` *(sobrescribir). Esto indica al compilador que el método ya existe en la clase padre y queremos reemplazarlo. Es opcional, aunque conviene incluirlo para evitar errores.*

Lo primero que hay que hacer al sobrescribir un método suele ser llamar al método de la clase de la que hemos heredado. Para referirnos a nuestra clase padre usaremos la palabra reservada `super`. El método termina indicando que la actividad va a visualizarse en una determinada vista. Esta vista está definida en los recursos. Más adelante se describe la finalidad de cada fichero y carpeta de este proyecto.


Vídeo[tutorial]: *Un primer proyecto Android*

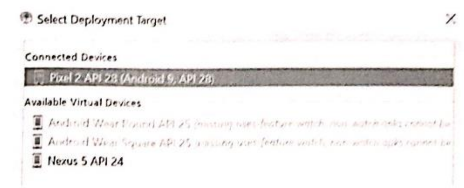
1.8. Ejecución del programa

Una vez creada esta primera aplicación, vamos a ver dos alternativas para ejecutarla: en un emulador y en un dispositivo real.

1.8.1. Ejecución en el emulador


Ejercicio: *Ejecución en el emulador*

1. Selecciona *Run > Run 'app'* (*Mayús-F10*) o pulsa el icono ▶ de la barra de herramientas.
2. Te preguntará sobre que dispositivo quieres ejecutar la aplicación:



Te permite escoger entre dispositivos conectados (AVD o reales), lanzar un AVD ya creado o crear uno nuevo.

3. Una vez que el emulador esté cargado, debes ver algo así:



1.8.2. Ejecución en un terminal real

También es posible ejecutar y depurar tus programas en un terminal real. Incluso es una opción más rápida y fiable que utilizar un emulador. No tienes más que usar un cable USB para conectar el terminal al PC. Resulta imprescindible haber instalado un *driver* especial en el PC. Puedes encontrar un *driver* genérico que se encuentra en la carpeta de instalación del SDK *\sdk\extras\google\usb_driver*. Aunque lo más probable es que tengas que utilizar el *driver* del fabricante.



Ejercicio: *Ejecución en un terminal real*

1. Abre *Android SDK Manager* y asegúrate de que está instalado el paquete USB Driver. En caso contrario, instálalo.

SDK Platforms SDK Tools SDK Update Sites

Below are the available SDK developer tools. Once installed, Android Studio will automatically check for updates. Check "show package details" to display available versions of an SDK Tool.

Name	Version	Status
Android SDK Build-Tools 31-rc3		Update Available: 31.0.0
NDK (Side by side)		Not Installed
Android SDK Command-line Tools (latest)		Not Installed
CMake		Not Installed
Android Auto API Simulators	1	Not installed
Android Auto Desktop Head Unit Emulator	2.0.0 rc1	Not installed
Android Emulator	29.3.4	Update Available: 30.5.5
Android Emulator Hypervisor Driver for AMD Processors (installer)	1.3.0	Update Available: 1.7.0
Android SDK Platform-Tools	29.0.5	Update Available: 31.0.2
Android SDK Tools	26.1.1	Installed
Google Play APK Expansion library	1	Not installed
Google Play Instant Development SDK	1.9.0	Not installed
Google Play Licensing Library	1	Not installed
Google Play services	49	Not installed
Google USB Driver	13	Installed
Google Web Driver	2	Not installed
Intel x86 Emulator Accelerator (HAXM installer)	7.5.4	Update Available: 7.6.5
Layout Inspector image server for API 29-30	6	Not installed

2. Posiblemente, este *driver* genérico no sea adecuado para tu terminal y tengas que utilizar el del fabricante. Si no dispones de él, puedes buscarlo en:

   http://developer.android.com/tools/extras/oem-usb.html

3. A partir de Android 4.2 las opciones para desarrolladores vienen ocultas por defecto. De esta forma, un usuario sin experiencia no podrá activar estas opciones de forma accidental. Para activar las opciones de desarrollo tienes que ir a Ajustes > Información del teléfono y pulsar siete veces sobre el número de compilación. Tras esto aparecerá el mensaje "¡Ahora eres un desarrollador!" y nos mostrará más ajustes.

4. En el terminal accede al menú *Ajustes > Opciones de desarrollador* y asegúrate de que la opción *Depuración de USB* está activada.

Depuración

Depuración por USB
Activar el modo de depuración cuando el dispositivo esté conectado por USB

5. Conecta el cable USB.

6. Se indicará que hay un nuevo *hardware* y te pedirá que le indiques el controlador.

   NOTA: *En Windows, si indicas un controlador incorrecto no funcionará. Además, la próxima vez que conectes el cable no te pedirá la instalación del controlador. Para desinstalar el controlador sigue los siguientes pasos:*

   1. *Asegúrate de haber desinstalado el controlador incorrecto.*
   2. *Accede al registro del sistema (*Inicio > ejecutar > RegEdit*). Busca la siguiente clave y bórrala: "vid_0bb4&pid_0c02".*
   3. *Vuelve al paso 3 del ejercicio.*

7. Selecciona de nuevo *Run > Run 'app'* (*Mayús-F10*) o pulsa el icono ▶. Aparecerá una ventana que te permite escoger en qué dispositivo o emulador quieres ejecutar la aplicación.

8. Selecciona el dispositivo real y pulsa *OK*.

1.9. Ficheros y carpetas de un proyecto Android

Lo primero que conviene que conozcas es que un proyecto en Android Studio puede contener varios módulos. Cada módulo corresponde a una aplicación o ejecutable diferente. Disponer de varios módulos en un mismo proyecto nos será muy útil cuando queramos crear varias versiones de nuestra aplicación, para dispositivo móvil, Wear, TV, Things, etc. También si queremos crear varias versiones de nuestra aplicación con nivel mínimo de SDK diferentes. En este libro solo vamos a desarrollar aplicaciones para móviles, por lo que no vamos a necesitar un módulo. Este módulo ha sido creado con el nombre *app*.

Cada módulo en Android está formado por un descriptor de la aplicación (*manifests*), el código fuente en Java (*java*), una serie de ficheros con recursos (*res*) y ficheros para construir el módulo (*Gradle Scripts*). Cada elemento se almacena en una carpeta específica, que hemos indicado entre paréntesis. Aprovecharemos el proyecto que acabamos de crear para estudiar la estructura de un proyecto en Android Studio. No te asustes con el exceso de información. Más adelante se dará más detalles sobre la finalidad de cada fichero.

AndroidManifest.xml: Este fichero describe la aplicación Android. Se define su nombre, paquete, icono, estilos, etc. Se indican las *actividades*, las *intenciones*, los *servicios* y los *proveedores de contenido* de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica el paquete Java, la versión de la aplicación, etc.

app
manifests
java
com.example.myapplication
MainActivity
com.example.myapplication
ExampleInstrumentedTest
com.example.myapplication
ExampleUnitTest
res
drawable
layout
activity_main.xml
mipmap
values
colors.xml
strings.xml
themes (2)
Gradle Scripts
build.gradle
build.gradle
gradle-wrapper.properties
proguard-rules.pro
gradle.properties
settings.gradle
local.properties (SDK Location)

java: Carpeta que contiene el código fuente de la aplicación. Como puedes observar los ficheros Java se almacenan en carpetas según el nombre de su paquete.

MainActivity: Clase con el código de la actividad inicial.

ExampleInstrumentTest: Clase para insertar código de testeo de la aplicación.

ExampleUnitTest: Clase para insertar test unitarios sobre otras clases.

res: Carpeta que contiene los recursos usados por la aplicación.

drawable: En esta carpeta se almacenan los ficheros de imágenes (JPG o PNG) y descriptores de imágenes en XML.

mipmap: En una carpeta guardaremos el icono de la aplicación. En el proyecto se ha incluido el fichero *ic_launcher.png* que será utilizado como icono de la aplicación. Observa cómo este recurso se ha añadido en seis versiones diferentes. Como veremos en el siguiente capítulo, usaremos un sufijo especial si queremos tener varias versiones de un recurso, de forma que solo se cargue al cumplirse una determinada condición. Por ejemplo: `(hdpi)` significa que solo ha de cargar los recursos contenidos en esta carpeta cuando el dispositivo donde se instala la aplicación tenga una densidad gráfica alta (180- dpi); `(mdpi)` se utilizará con densidad gráfica alta (180- dpi). Si pulsas sobre las diferentes versiones del recurso, observarás como se trata del mismo icono, pero con más o menos resolución de forma que, en función de la densidad gráfica del dispositivo, se ocupe un tamaño similar en la pantalla. El fichero *ic_launcher_round.png* es similar, pero se utiliza cuando se quieren usar iconos redondos.

layout: Contiene ficheros XML con vistas de la aplicación. Las vistas nos permitirán configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación. Se utiliza un formato similar al HTML usado para diseñar páginas web. Se tratarán en el siguiente capítulo.

menu: Ficheros XML con los menús de cada actividad. En el proyecto no hay ningún menú por lo que no se muestra esta carpeta.

values: También utilizaremos ficheros XML para indicar valores usados en la aplicación, de esta manera podremos cambiarlos desde estos ficheros sin necesidad de ir al código fuente. En *colors.xml* se definen los tres colores primarios de la aplicación. En *dimens.xml* se pueden definir dimensiones como el margen por defecto o el ancho de los botones. En el fichero *strings.xml*, tendrás que definir todas las cadenas de caracteres de tu aplicación. Creando recursos alternativos resultará muy sencillo traducir una aplicación a otro idioma. Finalmente, en *themes.xml*, podrás definir los estilos y temas de tu aplicación. Se estudian en el siguiente capítulo.

anim: Contiene ficheros XML con animaciones de vistas (Tween). Las animaciones se describen al final del capítulo 4.

animator: Contiene ficheros XML con animaciones de propiedades.

xml: Otros ficheros XML requeridos por la aplicación.

raw: Ficheros adicionales que no se encuentran en formato XML.

Gradle Scripts: En esta carpeta se almacenan una serie de ficheros Gradle que permiten compilar y construir la aplicación. Observa como algunos hacen referencia al módulo app y el resto son para configurar todo el proyecto. El fichero más importante es *build.gradle (Module:app)* que es donde se configuran las opciones de compilación del módulo:

```
plugins { id 'com.android.application'}
android {
    compileSdkVersion 30
    buildToolsVersion "30.0.2"
    defaultConfig {
       applicationId "com.example.myapplication"
       minSdkVersion 19
       targetSdkVersion 30
       versionCode 1
       versionName "1.0"
       testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
       release {
           minifyEnabled false
           proguardFiles getDefaultProguardFile('proguard-android-optimi…
       }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
```

```
    }
}
dependencies {
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.3.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
}
```

El primer parámetro que podemos configurar es `compileSdkVersion` que nos permite definir la versión del sdk con la que compilamos la aplicación. Las nuevas versiones no solo añaden funcionalidades al API, también añaden mejoras en los procesos. Por ejemplo, a partir de la versión 3.0 (API 11) solo se permite el acceso a Internet desde un hilo auxiliar[7]. `applicationId` suele coincidir con el nombre del paquete Java creado para la aplicación. Se utiliza como identificador único de la aplicación, de forma que no se permite instalar una aplicación si ya existe otra con el mismo id. `minSdkVersion` especifica el nivel mínimo de API que requiere la aplicación. Es un parámetro de gran importancia, la aplicación no podrá ser instalada en dispositivos con versiones anteriores y solo podremos usar las funcionalidades del API hasta este nivel (con excepción de las librerías de compatibilidad). `targetSdkVersion` indica la versión más alta con la que se ha puesto a prueba la aplicación. Cuando salgan nuevas versiones del SDK tendrás que comprobar la aplicación con estas versiones y actualizar el valor. `versionCode` y `versionName` indica la versión de tu aplicación. Cada vez que publiques una nueva versión incrementa en uno el valor de `versionCode` y aumenta el valor de `versionName` según la importancia de la actualización. Si es una actualización menor el nuevo valor podría ser `"1.1"` y si es mayor `"2.0"`.

Dentro de `buildTypes` se añaden otras configuraciones dependiendo del tipo de compilación que queramos (`relase` para distribución, `debug` para depuración, etc.). Los comandos que aparecen configuran la ofuscación de código. Para más información leer el capítulo «Ingeniería Inversa en Android» de El *Gran Libro de Android Avanzado*.

Un apartado importante es el de `dependencies`. En él has de indicar todas las librerías que han de ser incluidas en nuestro proyecto. Si necesitas usar alguna librería de compatibilidad adicional has de incluirla aquí.



Preguntas de repaso: *Elementos de un proyecto*

1.10. Componentes de una aplicación

Existe una serie de elementos clave que resultan imprescindibles para desarrollar aplicaciones en Android. En este apartado vamos a realizar una descripción inicial

[7] Se describe con detalle en el capítulo 10.

de algunos de los más importantes. A lo largo del libro se describirán con más detalle las clases Java que implementan cada uno de estos componentes.

1.10.1. Vista (View)

Las *vistas* son los elementos que componen la interfaz de usuario de una aplicación: por ejemplo, un botón o una entrada de texto. Todas las vistas van a ser objetos descendientes de la clase `View` y, por tanto, pueden ser definidas utilizando código Java. Sin embargo, lo habitual será definir las vistas utilizando un fichero XML y dejar que el sistema cree los objetos por nosotros a partir de este fichero. Esta forma de trabajar es muy similar a la definición de una página web utilizando código HTML.

1.10.2. *Layout*

Un *layout* es un conjunto de vistas agrupadas de una determinada forma. Vamos a disponer de diferentes tipos de *layouts* para organizar las vistas de forma lineal, en cuadrícula o indicando la posición absoluta de cada vista. Los *layouts* también son objetos descendientes de la clase View. Igual que las vistas, los *layouts* pueden ser definidos en código, aunque la forma habitual de definirlos es utilizando código XML.

1.10.3. Actividad (*Activity*)

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, coloquialmente conocidos como pantallas de la aplicación. En Android cada uno de estos elementos, o pantallas, se conoce como *actividad*. Su función principal es la creación de la interfaz de usuario. Una aplicación suele necesitar varias actividades para crear la interfaz de usuario. Las diferentes actividades creadas serán independientes entre sí, aunque todas trabajarán para un objetivo común. Una actividad se define en una clase descendiente de `Activity` y utiliza un layout para que define su apariencia.

1.10.4. Fragmentos (*Fragment*)

La llegada de las tabletas trajo el problema de que las aplicaciones de Android ahora deben soportar pantallas más grandes. Si diseñamos una aplicación pensada para un dispositivo móvil y luego la ejecutamos en una tableta, el resultado no suele resultar satisfactorio.

Para ayudar al diseñador a resolver este problema, en la versión 3.0 de Android aparecen los *fragments*. Un *fragment* está formado por la unión de varias vistas para crear un bloque funcional de la interfaz de usuario. Una vez creados los *fragments*, podemos combinar uno o varios *fragments* dentro de una actividad, según el tamaño de pantalla disponible.



Vídeo[tutorial]: *Los fragments en Android*

El uso de *fragments* puede ser algo complejo, por lo que recomendamos dominar primero conceptos como *actividad*, *vista* y *layout* antes de abordar su

aprendizaje. No obstante, es un concepto importante en Android y todo programador en esta plataforma ha de saber utilizarlos. Véase el anexo A para aprender más sobre *fragments*.

1.10.5. Servicio (*Service*)

Un *servicio* es un proceso que se ejecuta "detrás", sin la necesidad de una interacción con el usuario. Es algo parecido a un *demonio* en Unix o a un *servicio* en Windows. Se utilizan cuando queramos tener en ejecución un código de manera continua, aunque el usuario cambie de actividad. En Android disponemos de dos tipos de servicios: servicios locales, que son ejecutados en el mismo proceso, y servicios remotos, que son ejecutados en procesos separados. Los servicios se estudian en el capítulo 8.

1.10.6. Intención (*Intent*)

Una intención representa la voluntad de realizar alguna acción, como realizar una llamada de teléfono o visualizar una página web. Se utiliza cada vez que queramos:

- Lanzar una *actividad*
- Lanzar un *servicio*
- Enviar un anuncio broadcast
- Comunicarnos con un servicio

Los componentes lanzados pueden ser internos o externos a nuestra aplicación. También utilizaremos las intenciones para el intercambio de información entre estos componentes.

1.10.7. Receptor de anuncios (*Broadcast Receiver*)

Un *receptor de anuncios* recibe anuncios *broadcast* y reacciona ante ellos. Los anuncios *broadcast* pueden ser originados por el sistema (por ejemplo: *Batería baja*, *Llamada entrante*) o por las aplicaciones. Las aplicaciones también pueden crear y lanzar nuevos tipos de anuncios *broadcast*. Los receptores de anuncios no disponen de interfaz de usuario, aunque pueden iniciar una actividad si lo estiman oportuno. Los receptores de anuncios se estudian en el CAPÍTULO 8.

1.10.8. Proveedores de contenido (*Content Provider*)

En muchas ocasiones, las aplicaciones instaladas en un terminal Android necesitan compartir información. Android define un mecanismo estándar para que las aplicaciones puedan compartir datos sin necesidad de comprometer la seguridad del sistema de ficheros. Con este mecanismo podremos acceder a datos de otras aplicaciones, como la lista de contactos, o proporcionar datos a otras aplicaciones. Los ContentProvider se estudian en el capítulo 9.



Preguntas de repaso: *Componentes de una aplicación*

1.11. Documentación y aplicaciones de ejemplo

Aunque en este libro vas a aprender mucho, resultaría imposible tocar todos los aspectos de Android y con un elevado nivel de profundidad. Por lo tanto, resulta imprescindible que dispongas de fuentes de información para consultar los aspectos que vayas necesitando. En este apartado te proponemos dos alternativas: el acceso a documentación sobre Android y el estudio de ejemplos.

1.11.1. Dónde encontrar documentación

Puedes encontrar una completa documentación del SDK localmente en:

http://developer.android.com/

Se incluye la descripción de todas las clases (*Develop > Reference*), conceptos clave y otros tipos de recursos.

Muchos de los recursos utilizados en este libro puedes encontrarlos en:

http://www.androidcurso.com/

Para resolver dudas puntuales sobre programación te recomendamos la siguiente web de preguntas y respuestas:

http://stackoverflow.com/

1.11.2. Repositorio de ejemplos en GitHub

Otra opción muy interesante para aprender nuevos aspectos de programación consiste en estudiar ejemplos. Google ha preparado un repositorio de ejemplos en *GitHub* que pueden ser instalados desde **Android Studio**.

Ejercicio: *Instalación de un ejemplo desde GitHub*

1. Selecciona *File > New > Import Sample...* Aparecerá la siguiente ventana:

Select a sample to import

Notifications
Permissions
Renderscript
Basic Render Script
Render Script Intrinsic
Security
Fingerprint Dialog (Kotlin)
Wear Notifications
MESSAGING STYLE

Los proyectos se encuentran clasificados en categorías: *Admin*, *Background*, *Connectivity*, *Content*, *Input*, *Media*, *Notification*, … Selecciona un proyecto de alguna de estas categorías. A la derecha podrás leer una breve descripción o ver una vista previa.

2. Pulsa *Next* para pasar a la siguiente ventana. Podrás configurar el nombre de la aplicación, explorar el proyecto accediendo a su sitio web en GitHub e indicar la carpeta donde quieres descargarlo.

Provide information about your project

Application name: Notifications

3. Pulsa *Finish* y a continuación ejecuta el proyecto seleccionado.
4. Estudia el código del proyecto.

1.12. Depurar

Programación y errores de código son un binomio inseparable. Por lo tanto, resulta fundamental sacar el máximo provecho a las herramientas de depuración.

1.12.1. Depurar con el entorno de desarrollo

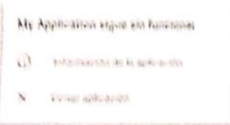
Android Studio integra excelentes herramientas para la depuración de código. Para probarlas, introduce un error en tu código modificando MainActivity de forma que en método onCreate() tenga este código:

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   Object o = null;
   o.toString();
   setContentView(R.layout.activity_main);
}
```

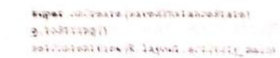
```
lateinit var o: Any

override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   o.toString()
   setContentView(R.layout.activity_main)
}
```

Este cambio introduce en *Java* un NullPointerException y en *Kotlin* un UninitializedPropertyAccessException. Si ahora ejecutas tu aplicación, te aparecerá algo similar a:



Pulsa *Cerrar* para finalizar la aplicación. Para averiguar más sobre el error, inserta un punto de ruptura (*breakpoint*) en el código fuente en la línea o.toString() (el *breakpoint* se introduce haciendo clic en la barra de la izquierda).



Entonces selecciona *Run > Debug 'app'* (Mayús+F9) o pulse en para ejecutarlo en modo *Debug*. Tu aplicación se reiniciará mostrando el siguiente mensaje:

Waiting For Debugger

Pero esta vez quedará suspendida cuando alcance el punto de ruptura que has introducido. Entonces puedes recorrer el código en modo *Debug*, igual que se haría en cualquier otro entorno de programación. Pulsa en *Run > Step Over* (F8) para ir ejecutando las líneas una a una.



Vídeo[tutorial]: *Depurar con Android Studio*

1.12.2. Depurar con mensajes Log

El sistema Android utiliza el fichero *LogCat* para registrar todos los problemas y eventos principales que ocurren en el sistema. Ante cualquier error resulta muy interesante consultarlo para tratar de encontrar su origen.

La clase Log proporciona un mecanismo para introducir mensajes desde nuestro código en este fichero. Puede ser muy útil para depurar nuestros programas o para verificar el funcionamiento del código. Disponemos de varios métodos para generar distintos tipos de mensajes:

Log.e(): Errors
Log.w(): Warnings
Log.i(): Information
Log.d(): Debugging
Log.v(): Verbose

Ejercicio: *Depurar con mensajes Log*

1. Modifica la clase `MainActivity` introduciendo la línea que aparece subrayada:

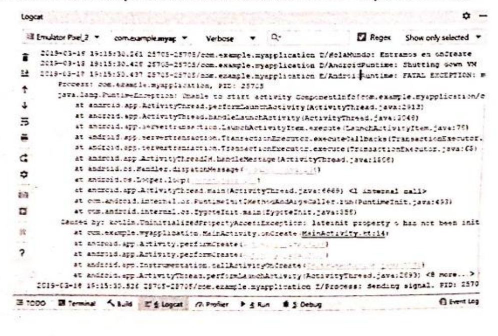
```
@Override
public void onCreate(Bundle savedInstanceState) {
   Log.d("HolaMundo","Entramos en onCreate");
   super.onCreate(savedInstanceState);
   Object o = null;
   o.toString();
   setContentView(R.layout.activity_main);
}
```

```
lateinit var o: Any

override fun onCreate(savedInstanceState: Bundle?) {
   Log.d("HolaMundo","Entramos en onCreate");
   super.onCreate(savedInstanceState)
   o.toString()
   setContentView(R.layout.activity_main)
}
```

Nota sobre Java/Kotlin: *Para poder utilizar la clase* `Log` *has de importar un nuevo paquete. Para ello añade al principio* `import android.util.Log;` *Otra alternativa es pulsar* **Alt-Intro** *para que se añadan automáticamente los paquetes que faltan. En algunos casos, el sistema puede encontrar dos paquetes con la clase* `Log`, *y puede tener dudas sobre cual importar. En estos casos te preguntará.*

2. Ejecuta la aplicación. Aparecerá un error.
3. En **Android Studio** aparecerá automáticamente en la parte inferior:



En la primera línea de la captura anterior, comprobamos que se pudo entrar dentro de `onCreate()`. Dos líneas más abajo se indica una excepción. La información mostrada suele ser excesiva. Te recomendamos que busques las palabras "Caused by" para ver el tipo de excepción y la primera referencia a un paquete escrito por nosotros, "com.example.jtomas.myapplication". En este ejemplo, las líneas clave son: **en Java** *"Cased by: java.lang.NullPointerException at conm.example.jtomas.myapplication.MainActivity.onCreate(MainActivity.java:17)"*. **En Kotlin** *"Caused by: kotlin.UninitializedPropertyAccessException: lateinit property o has not been initialized at com.example.myapplication.MainActivity.onCreate (MainActivity.kt:14)"*

4. Haz clic en *(MainActivity.java:17)* o *(MainActivity.kt:14)*. Te abrirá la actividad *MainActivity* y te situará en la línea donde se ha producido el error.



Vídeo[tutorial]: *LogCat con Android Studio*

1.13. Introducción Java/Kotlin y aplicación Mis Lugares

Si no dominas el lenguaje de programación Java o Kotlin, puede ser una buena idea repasar los conceptos más importantes de uno de estos lenguajes antes de comenzar a realizar aplicaciones en Android. Existe una gran cantidad de libros[8] y tutoriales en Internet que pueden ayudarte en este propósito. También puede ser de utilidad una consulta al Anexo C.

Hemos preparado un conjunto de breves tutoriales que te mostrarán lo esencial de Java y Kotlin. Suponemos que ya tienes conocimientos de programación. De no ser así, puede que tengas dificultades en seguirlos.



Vídeo[tutorial]: *Características de Java*



Vídeo[tutorial]: *Creación y utilización de clases*[9]



Enlaces de interés: *Comentarios y documentación javadoc*

http://www.androidcurso.com/index.php/27

[8] Recomendamos: *Piensa en Java*, de Bruce Eckel, Ed.Prentice Hall y Kotlin for Android Developers de Antonio Leiva.

[9] Tutorial web: http://www.androidcurso.com/index.php/24 y http://www.androidcurso.com/index.php/25



Vídeo[tutorial]: *Encapsulamiento y visibilidad en Java*[10]

A lo largo de este libro vamos a crear un par de aplicaciones. Una de ellas será Mis Lugares, que nos permitirá recordar los lugares donde hemos estado o que más nos gustan. Tras realizar los tutoriales que aparecen en este apartado, dispondrás de varias clases que te serán de utilidad en la aplicación Mis Lugares (estas clases son: `Lugar`, `RepositorioLugares`, `TipoLugar` y `Geopunto`).



Vídeo[tutorial]: *La aplicación Mis Lugares*

Aunque ya tengas experiencia en Java o Kotlin, te recomendamos que realices los tutoriales que incluimos a continuación. De esta forma, podrás familiarizarte con las clases que usaremos en Mis Lugares.

1.13.1. La clase Lugar

La aplicación Mis Lugares permite gestionar una colección de lugares. Para cada lugar vamos a poder almacenar mucha información: nombre, dirección, posición geográfica, etc. El primer paso a realizar va a ser crear una clase que nos permita trabajar con este tipo de información. Este tipo de clase se conoce muchas veces como POJO o clase de datos.



Ejercicio: *Creación de la clase Lugar en Android Studio*

Android Studio está pensado exclusivamente para crear aplicaciones Android. Sin embargo, si sigues los siguientes pasos podrás crear una aplicación 100 % Java o Kotlin.

1. Crea un nuevo proyecto (*File > New > New Project...*) con los siguientes datos:

```
Phone and Tablet / Add No Activity
Name: Mis Lugares Java ó Mis Lugares Kotlin
Package name: com.example.mislugares
Language: Java ó Kotlin
Minimum API level: API 19 Android 4.4 (KitKat)
```

NOTA: Deja el resto de los parámetros con su valor por defecto.

2. Pulsa en *File > New > New Module*. Selecciona *Java Library* y pulsa *Next*.

[10] Tutorial web: http://www.androidcurso.com/index.php/32

3. Introduce en *Library name:* `MisLugares`, como *Java pakage name:* `com.example.mislugares` y en *Java class name:* `Lugar`. Pulsa el botón *Finish*. Se creará un nuevo módulo Java dentro de tu proyecto Android.
4. **Para Kotlin**, en el explorador de proyecto busca la clase Java y, con el botón derecho, selecciona *Convert Java File to Kotlin File*.
5. Reemplaza el código de la clase `Lugar` por el siguiente:

```
package com.example.mislugares;

public class Lugar {
    private String nombre;
    private String direccion;
    private GeoPunto posicion;
    private String foto;
    private int telefono;
    private String url;
    private String comentario;
    private long fecha;
    private float valoracion;

    public Lugar(String nombre, String direccion, double longitud,
        double latitud, int telefono, String url, String comentario,
        int valoracion) {
        fecha = System.currentTimeMillis();
        posicion = new GeoPunto(longitud, latitud);
        this.nombre = nombre;
        this.direccion = direccion;
        this.telefono = telefono;
        this.url = url;
        this.comentario = comentario;
        this.valoracion = valoracion;
    }
}
```

```
package com.example.mislugares

data class Lugar(val nombre: String,
                 var direccion: String = "",
                 var posicion: GeoPunto = GeoPunto.SIN_POSICION,
                 var foto: String = "",
                 var telefono: Int = 0,
                 var url: String= "",
                 var comentarios: String = "",
                 var fecha: Long = System.currentTimeMillis(),
                 var valoracion: Float = 3.5F
)
```

Para Java observa cómo se definen los atributos de la clase y como en el constructor se inicializa para un objeto concreto según los parámetros indicados. En estos parámetros no se indica el atributo `fecha`. Este representa el día y la hora en que visitamos ese lugar por última vez. Se codifica mediante un `long` (número entero de 64 bits), que supondremos en formato *Epoch time* o tiempo

Unix[11]. Es decir, número de milisegundos transcurridos desde 1970. El método `System.currentTimeMillis()` nos devuelve la fecha y la hora actuales en este formato. Por lo tanto, siempre que usemos este constructor, en `fecha` se almacenará el instante en que el objeto fue creado.

Para Kotlin en el constructor principal indicamos directamente los atributos de la clase y en algunos casos los valores por defecto. Los *getters* y *setters* son creados automáticamente. Además, al haber indicado `data class` se crean otras funciones como `toString()`. Por lo tanto, podrás saltarte los siguientes dos puntos.

6. **Solo para Java**, crea los métodos *getters* y *setters* para acceder a todos los atributos de la clase. Solo tienes que pulsar con el botón derecho y seleccionar la opción *Generate… > Getter and Setter* y selecciona todos los atributos mientras mantienes pulsada la tecla *Ctrl*.

7. **Solo para Java**, pulsa con el botón derecho sobre el código y selecciona la opción *Generate… > toString()*. Selecciona todos los atributos y pulsa *OK*. Se añadirá un método similar a:

```
@Override public String toString() {
   return "Lugar {nombre=" + nombre + ", direccion=" + direccion
      + ", posicion=" + posicion + ", foto=" + foto + ", telefono="
      + telefono + ", url=" + url + ", comentario=" + comentario
      + ", fecha=" + fecha + ", valoracion=" + valoracion + "}";
}
```

***NOTA:** El significado de `@Override` se explica más adelante.*

8. Dentro del explorador del módulo *MisLugares / java / com.example.mislugares* pulsa con el botón derecho y selecciona *New > Java Class o Kotlin File/Class.*

9. Introduce en el campo *Name*: `GeoPunto` y pulsa *Ok*. Reemplaza el código por el siguiente (dejando la línea del `package`):

```
public class GeoPunto {

   private double longitud, latitud;

   static public GeoPunto SIN_POSICION = new GeoPunto(0.0,0.0);

   public GeoPunto(double longitud, double latitud) {
      this.longitud= longitud;
      this.latitud= latitud;
   }
   public String toString() {
      return new String("longitud:" + longitud + ", latitud:"+ latitud);
   }
   ...lic double distancia(GeoPunto punto) {
      ...DIO_TIERRA = 6371000; // en metros
      ...itud - punto.latitud);
      ...ngitud);
```



```
      double lat2 = Math.toRadians(latitud);
      double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                 Math.sin(dLon/2) * Math.sin(dLon/2) *
                 Math.cos(lat1) * Math.cos(lat2);
      double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
      return c * RADIO_TIERRA;
   }
}
```

```
data class GeoPunto(var longitud: Double, var latitud: Double) {

   companion object {
      val SIN_POSICION = GeoPunto(0.0,0.0)
   }

   fun distancia(punto: GeoPunto): Double {
      val RADIO_TIERRA = 6371000.0 // en metros
      val dLat = Math.toRadians(latitud - punto.latitud)
      val dLon = Math.toRadians(longitud - punto.longitud)
      val lat1 = Math.toRadians(punto.latitud)
      val lat2 = Math.toRadians(latitud)
      val a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
              Math.sin(dLon / 2) * Math.sin(dLon / 2) *
              Math.cos(lat1) * Math.cos(lat2)
      val c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))
      return c * RADIO_TIERRA
   }
}
```

El objeto `SIN_POSICION` será utilizado cuando se quiera indicar que un lugar no tiene posición asignada. Observa que es un objeto de tipo estático. En Java se indica con `static` y en Kotlin con `companion object`. Esto significa que solo va a haber una instancia de este objeto creada desde el principio. Para acceder a ella usaremos `GeoPunto.SIN_POSICION`.

10. Solo **para Java**, crea en esta clase los métodos *getters* y *setters* para acceder a los dos atributos. Igual que antes, pulsa con el botón derecho y seleccionar la opción *Generate… > Getter and Setter*. Realiza la misma operación para *equals() and hasCode()*.

11. **Para Java y Kotlin**, crea una nueva clase Java con nombre: `Principal`. Android Studio no permite que la clase principal esté en Kotlin.

12. Reemplaza el código por el mostrado (dejando la línea del `package`):

```
class Principal {
   public static void main(String[] main) {
     Lugar lugar = new Lugar("Escuela Politécnica Superior de Gandía",
        "C/ Paranimf, 1 46730 Gandia (SPAIN)", -0.166093, 38.995656,
        962849300, "http://www.epsg.upv.es",
        "Uno de los mejores lugares para formarse.", 3);
     System.out.println("Lugar " + lugar.toString());
```

Unix[11]. Es decir, número de milisegundos transcurridos desde 1970. El método `System.currentTimeMillis()` nos devuelve la fecha y la hora actuales en este formato. Por lo tanto, siempre que usemos este constructor, en `fecha` se almacenará el instante en que el objeto fue creado.

Para Kotlin en el constructor principal indicamos directamente los atributos de la clase y en algunos casos los valores por defecto. Los *getters* y *setters* son creados automáticamente. Además, al haber indicado `data class` se crean otras funciones como `toString()`. Por lo tanto, podrás saltarte los siguientes dos puntos.

6. **Solo para Java**, crea los métodos *getters* y *setters* para acceder a todos los atributos de la clase. Solo tienes que pulsar con el botón derecho y seleccionar la opción *Generate... > Getter and Setter* y selecciona todos los atributos mientras mantienes pulsada la tecla *Ctrl*.
7. **Solo para Java**, pulsa con el botón derecho sobre el código y selecciona la opción *Generate... > toString()*. Selecciona todos los atributos y pulsa *OK*. Se añadirá un método similar a:

```
@Override public String toString() {
   return "Lugar {nombre=" + nombre + ", direccion=" + direccion
      + ", posicion=" + posicion + ", foto=" + foto + ", telefono="
      + telefono + ", url=" + url + ", comentario=" + comentario
      + ", fecha=" + fecha + ", valoracion=" + valoracion + "}";
}
```

NOTA: El significado de @Override se explica más adelante.

8. Dentro del explorador del módulo *MisLugares / java / com.example.mislugares* pulsa con el botón derecho y selecciona *New > Java Class o Kotlin File/Class*.
9. Introduce en el campo *Name*: `GeoPunto` y pulsa *Ok*. Reemplaza el código por el siguiente (dejando la línea del `package`):

```
public class GeoPunto {

   private double longitud, latitud;

   static public GeoPunto SIN_POSICION = new GeoPunto(0.0,0.0);

   public GeoPunto(double longitud, double latitud) {
      this.longitud= longitud;
      this.latitud= latitud;
   }
    public String toString() {
      return new String("longitud:" + longitud + ", latitud:"+ latitud);
    }
    public double distancia(GeoPunto punto) {
         final double RADIO_TIERRA = 6371000; // en metros
         double dLat = Math.toRadians(latitud - punto.latitud);
         double dLon = Math.toRadians(longitud - punto.longitud);
         double lat1 = Math.toRadians(punto.latitud);
         double lat2 = Math.toRadians(latitud);
         double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
                    Math.sin(dLon/2) * Math.sin(dLon/2) *
                    Math.cos(lat1) * Math.cos(lat2);
         double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
         return c * RADIO_TIERRA;
    }
}
```

[11] http://es.wikipedia.org/wiki/Tiempo_Unix


```
data class GeoPunto(var longitud: Double, var latitud: Double) {

   companion object {
      val SIN_POSICION = GeoPunto(0.0,0.0)
   }

   fun distancia(punto: GeoPunto): Double {
      val RADIO_TIERRA = 6371000.0 // en metros
      val dLat = Math.toRadians(latitud - punto.latitud)
      val dLon = Math.toRadians(longitud - punto.longitud)
      val lat1 = Math.toRadians(punto.latitud)
      val lat2 = Math.toRadians(latitud)
      val a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
             Math.sin(dLon / 2) * Math.sin(dLon / 2) *
             Math.cos(lat1) * Math.cos(lat2)
      val c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a))
      return c * RADIO_TIERRA
   }
}
```

El objeto `SIN_POSICION` será utilizado cuando se quiera indicar que un lugar no tiene posición asignada. Observa que es un objeto de tipo estático. En Java se indica con `static` y en Kotlin con `companion object`. Esto significa que solo va a haber una instancia de este objeto creada desde el principio. Para acceder a ella usaremos `GeoPunto.SIN_POSICION`.

10. Solo **para Java**, crea en esta clase los métodos *getters* y *setters* para acceder a los dos atributos. Igual que antes, pulsa con el botón derecho y seleccionar la opción *Generate... > Getter and Setter*. Realiza la misma operación para *equals() and hasCode()*.
11. **Para Java y Kotlin**, crea una nueva clase Java con nombre: `Principal`. Android Studio no permite que la clase principal esté en Kotlin.
12. Reemplaza el código por el mostrado (dejando la línea del `package`):

```
class Principal {
   public static void main(String[] main) {
     Lugar lugar = new Lugar("Escuela Politécnica Superior de Gandía",
        "C/ Paranimf, 1 46730 Gandia (SPAIN)", -0.166093, 38.995656,
        962849300, "http://www.epsg.upv.es",
        "Uno de los mejores lugares para formarse.", 3);
     System.out.println("Lugar " + lugar.toString());
   }
}
```

```
class Principal {
   public static void main(String[] main) {
      Lugar lugar = new Lugar("Escuela Politécnica Superior de Gandía",
         "C/ Paranimf, 1 46730 Gandia (SPAIN)",
         new GeoPunto(-0.166093, 38.995656),
         962849300, "http://www.epsg.upv.es",
         "Uno de los mejores lugares para formarse.",
         System.currentTimeMillis(), 3);
      System.out.println("Lugar " + lugar.toString());
   }
}
```

La clase `Principal` es algo atípica: no tiene atributos ni constructor, únicamente el método `main`. Cuando en un proyecto existe una clase que tiene un método con este perfil, es el que se llama para comenzar la ejecución. Como parámetros, este método recibe un *array* de `Strings`. Esta información tiene interés cuando el programa se ejecuta desde la línea de comandos con parámetros.

Si trabajas con **Kotlin** has de usar un constructor diferente, dado que este tiene más parámetros.

13. Pulsa en el botón desplegable a la derecha del botón *Run* [app ▾] ▶. Selecciona *Edit Configurations...*
14. En la nueva ventana, haz clic en el signo + de la esquina superior izquierda y selecciona *Application*. Aparecerá una nueva configuración de aplicación. Selecciona en *Name:* `mislugares`, en *Main class:* `com.example.mislugares.Principal` y en *Use classpath of module:* `MisLugares`. Pulsa en *OK*.
15. Pulsa el botón *Ejecución* [mislugares ▾] ▶ y verifica que el resultado que aparece en la ventana de *Run* es similar a:

```
"C\Program ...
Lugar {nombre=Escuela Politécnica Superior de Gandía,
direccion=C/ Paranimf, 1 46730 Gandia (SPAIN),
posicion=longitud:-0.166093, latitud:38.995656, foto=null,
telefono=962849300, url=http://www.epsg.upv.es,
comentario=Uno de los mejores lugares para formarse.,
fecha=1392332854758, valoracion=3.0}

Process finished with exit code 0
```



Vídeo[tutorial]: *La Herencia en Java*[12]

[12] Tutorial web: http://www.androidcurso.com/index.php/29



Enlaces de interés: *Sobrecarga:*
http://www.androidcurso.com/index.php/30



Vídeo[tutorial]: *El polimorfismo en Java*[13] [14]

1.13.2. Tipos enumerados



Vídeo[tutorial]: *Tipos enumerados en Java*[15]



Vídeo[tutorial]: *Clases enumeradas en Kotlin*



Ejercicio: *El enumerado TipoLugar*

En este ejercicio vamos a crear un tipo enumerado para diferenciar entre diferentes tipos de establecimientos en la aplicación Mis Lugares. Además, a cada tipo de lugar le asociaremos un `String` con el nombre y un recurso gráfico.

1. Vamos a crear un nuevo tipo enumerado. Para ello pulsa con el botón derecho en el paquete `com.example.mislugares`. Selecciona *New > Java Class* e introduce en *Name*: `TipoLugar`, en *Kind:* `Enum` y pulsa *OK*.
2. Reemplaza el código por el siguiente (dejando la línea del `package`):

```
public enum TipoLugar {
    OTROS ("Otros", 5),
    RESTAURANTE ("Restaurante", 2),
    BAR ("Bar", 6),
    COPAS ("Copas", 0),
    ESPECTACULO ("Espectáculo", 0),
    HOTEL ("Hotel", 0),
```

[13] Tutorial web: http://www.androidcurso.com/index.php/31

[14] Tutorial web: http://www.androidcurso.com/index.php/31

[15] Tutorial web: http://www.androidcurso.com/index.php/461

```
    COMPRAS ("Compras", 0),
    EDUCACION ("Educación", 0),
    DEPORTE ("Deporte", 0),
    NATURALEZA ("Naturaleza", 0),
    GASOLINERA ("Gasolinera", 0);

    private final String texto;
    private final int recurso;

    TipoLugar(String texto, int recurso) {
        this.texto = texto;
        this.recurso = recurso;
    }

    public String getTexto() { return texto; }
    public int getRecurso() { return recurso; }
}
```

```
enum class TipoLugar private constructor(val texto:String, val recurso:Int){
    OTROS("Otros", 5),
    RESTAURANTE("Restaurante", 2),
    BAR("Bar", 6),
    COPAS("Copas", 0),
    ESPECTACULO("Espectáculo", 0),
    HOTEL("Hotel", 0),
    COMPRAS("Compras", 0),
    EDUCACION("Educación", 0),
    DEPORTE("Deporte", 0),
    NATURALEZA("Naturaleza", 0),
    GASOLINERA("Gasolinera", 0)
}
```

Si quieres puedes definir otros tipos de lugares para adaptar la aplicación a tus necesidades. Observa como a cada constante le asociamos un `String` con el nombre del tipo de lugar y un entero. El entero se utilizará más adelante para indicar un recurso gráfico en Android con un icono representativo del tipo.

3. Abre la clase `Lugar`. **En Kotlin** añade el código subrayado y salta al punto 8:

```
data class Lugar(… var posicion: GeoPunto,
                   var tipoLugar: TipoLugar, …
```

En Java añade el siguiente atributo a la clase:

```
private TipoLugar tipo;
```

4. Añade el parámetro `TipoLugar` en el constructor de la clase e inicializa el atributo anterior con este parámetro:

```
public Lugar(String nombre, String direccion, double longitud,
             double latitud, TipoLugar tipo, int telefono, String url,
             String comentario, int valoracion) {
    this.tipo = tipo;
    …
}
```


5. Añade los métodos *getter* y *setter* correspondientes. Para ello pulsa con el botón derecho y seleccionar la opción *Generate > Getter and Setter.*
6. Vamos a volver a generar el método `toString()`. Para ello pulsa con el botón derecho y seleccionar la opción *Generate > toString().* Pulsa *Yes* para reemplazar el método actual.
7. Abre la clase `Principal` y modifica la inicialización del objeto para que se incluya el nuevo parámetro, `TipoLugar.EDUCACION`, en el constructor.
8. Verifica el resultado ejecutando el proyecto.

1.13.3. Las colecciones I



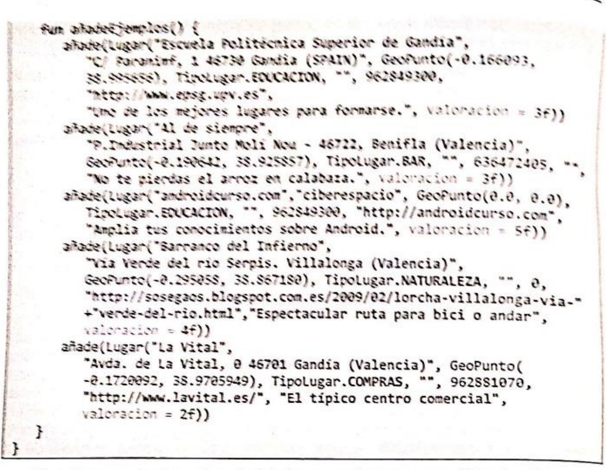
Ejercicio: *La interfaz RepositorioLugares*

En este ejercicio vamos a crear una interfaz que nos permita almacenar una lista de objetos *Lugar.* A lo largo del curso esta interfaz será implementada por dos clases. En esta unidad usaremos una lista almacenada en memoria y en la última unidad una base de datos. Usar esta interface nos va a permitir desacoplar la forma en la que almacenamos los datos del resto de la aplicación. Por ejemplo, si en un futuro queremos que los datos se almacenen en la nube, solo será necesario cambiar la implementación de esta interface, dejando idéntica el resto de la aplicación.

1. Dentro del explorador del proyecto *mislugares / java / com.example.mislugares*, pulsa con el botón derecho y selecciona *New > Java Class* o *New > Kotlin File/Class.*
2. Introduce en el campo *Name*: `RepositorioLugares` y en *Kind:* `Interface`.
3. Reemplaza el código por el siguiente (dejando la línea del `package`):

```
public interface RepositorioLugares {
    Lugar elemento(int id); //Devuelve el elemento dado su id
    void añade(Lugar lugar); //Añade el elemento indicado
    int nuevo(); //Añade un elemento en blanco y devuelve su id
    void borrar(int id); //Elimina el elemento con el id indicado
    int tamaño(); //Devuelve el número de elementos
    void actualiza(int id, Lugar lugar); //Reemplaza un elemento
}
```

```
interface RepositorioLugares {
    fun elemento(id: Int): Lugar  //Devuelve el elemento dado su id
    fun añade(lugar: Lugar)       //Añade el elemento indicado
    fun nuevo(): Int        //Añade un elemento en blanco y devuelve su id
    fun borrar(id: Int)     //Elimina el elemento con el id indicado
    fun tamaño(): Int       //Devuelve el número de elementos
    fun actualiza(id: Int, lugar: Lugar)  //Reemplaza un elemento
```



```
   fun añadeEjemplos() {
      añade(Lugar("Escuela Politécnica Superior de Gandía",
         "C/ Paranimf, 1 46730 Gandia (SPAIN)", GeoPunto(-0.166093,
         38.995656), TipoLugar.EDUCACION, "", 962849300,
         "http://www.epsg.upv.es",
         "Uno de los mejores lugares para formarse.", valoracion = 3f))
      añade(Lugar("Al de siempre",
         "P.Industrial Junto Molí Nou - 46722, Benifla (Valencia)",
         GeoPunto(-0.190642, 38.925857), TipoLugar.BAR, "", 636472405, "",
         "No te pierdas el arroz en calabaza.", valoracion = 3f))
      añade(Lugar("androidcurso.com","ciberespacio", GeoPunto(0.0, 0.0),
         TipoLugar.EDUCACION, "", 962849300, "http://androidcurso.com",
         "Amplia tus conocimientos sobre Android.", valoracion = 5f))
      añade(Lugar("Barranco del Infierno",
         "Vía Verde del río Serpis. Villalonga (Valencia)",
         GeoPunto(-0.295058, 38.867180), TipoLugar.NATURALEZA, "", 0,
         "http://sosegaos.blogspot.com.es/2009/02/lorcha-villalonga-via-"
         +"verde-del-rio.html","Espectacular ruta para bici o andar",
         valoracion = 4f))
      añade(Lugar("La Vital",
         "Avda. de La Vital, 0 46701 Gandía (Valencia)", GeoPunto(
         -0.1720092, 38.9705949), TipoLugar.COMPRAS, "", 962881070,
         "http://www.lavital.es/", "El típico centro comercial",
         valoracion = 2f))
   }
}
```

Una clase que implemente esta interfaz va a almacenar una lista de objetos de tipo `Lugar`. Mediante los métodos indicados vamos a poder acceder y modificar esta lista. Una interfaz también puede tener funciones estáticas, como `añadeEjemplos()`. En Java solo está permitido con API mínima >24, por lo que lo añadiremos esta función en una clase no abstracta.

4. Esta interface será usada en uno de los siguientes apartados.

1.13.4. Las colecciones II


Vídeo[tutorial]: *Las colecciones en Java*[16]


Vídeo[tutorial]: *Colecciones en Kotlin: introducción*


Vídeo[tutorial]: *Colecciones en Kotlin: List, Set y Map*

[16] Tutorial web: http://www.androidcurso.com/index.php/462



Ejercicio: *La clase LugaresLista*

En este ejercicio vamos a crear la clase `LugaresLista`, que tiene como finalidad almacenar y gestionar un conjunto de objetos `Lugar` dentro de una lista.

1. Dentro del paquete `com.example.mislugares` añade la clase `LugaresLista` y reemplaza el código por el siguiente:

```
public class LugaresLista implements RepositorioLugares {
   protected List<Lugar> listaLugares;

   public LugaresLista() {
      listaLugares = new ArrayList<Lugar>();
      añadeEjemplos();
   }

   public Lugar elemento(int id) {
      return listaLugares.get(id);
   }

   public void añade(Lugar lugar) {
      listaLugares.add(lugar);
   }

   public int nuevo() {
      Lugar lugar = new Lugar();
      listaLugares.add(lugar);
      return listaLugares.size()-1;
   }

   public void borrar(int id) {
      listaLugares.remove(id);
   }

   public int tamaño() {
      return listaLugares.size();
   }

   public void actualiza(int id, Lugar lugar) {
      listaLugares.set(id, lugar);
   }

   public void añadeEjemplos() {
      añade(new Lugar("Escuela Politécnica Superior de Gandía",
         "C/ Paranimf, 1 46730 Gandia (SPAIN)", -0.166093, 38.995656,
         TipoLugar.EDUCACION, 962849300, "http://www.epsg.upv.es",
         "Uno de los mejores lugares para formarse.", 3));
      añade(new Lugar("Al de siempre",
         "P.Industrial Junto Molí Nou - 46722, Benifla (Valencia)",
         -0.190642, 38.925857, TipoLugar.BAR, 636472405, "",
         "No te pierdas el arroz en calabaza.", 3));
      añade(new Lugar("androidcurso.com",
         "ciberespacio", 0.0, 0.0, TipoLugar.EDUCACION,
```

```
      962849300, "http://androidcurso.com",
      "Amplía tus conocimientos sobre Android.", 5));
    añade(new Lugar("Barranco del Infierno",
      "Vía Verde del río Serpis. Villalonga (Valencia)",
      -0.295058, 38.867180, TipoLugar.NATURALEZA, 0,
      "http://sosegaos.blogspot.com.es/2009/02/lorcha-villalonga-via"+
      "-verde-del-rio.html","Espectacular ruta para bici o andar",4));
    añade(new Lugar("La Vital",
      "Avda. de La Vital, 0 46701 Gandía (Valencia)", -0.1720092,
      38.9705949, TipoLugar.COMPRAS, 962881070,
      "http://www.lavital.es/", "El típico centro comercial", 2));
  }
}
```

```
class LugaresLista : RepositorioLugares {
   val listaLugares= mutableListOf<Lugar>()

   override fun elemento(id: Int): Lugar {
      return listaLugares[id]
   }

   override fun añade(lugar: Lugar) {
      listaLugares.add(lugar)
   }

   override fun nuevo(): Int {
      val lugar = Lugar("Nuevo lugar")
      listaLugares.add(lugar)
      return listaLugares.size - 1
   }

   override fun borrar(id: Int) {
      listaLugares.removeAt(id)
   }

   override fun tamaño(): Int {
      return listaLugares.size
   }

   override fun actualiza(id: Int, lugar: Lugar) {
      listaLugares[id] = lugar
   }
}
```

2. Pulsa *Alt-Intro* para que se añadan los `import` de las clases utilizadas.

```
import java.util.ArrayList;
import java.util.List;
```

3. **Para Java**, añade la siguiente sobrecarga al constructor a la clase `Lugar`:

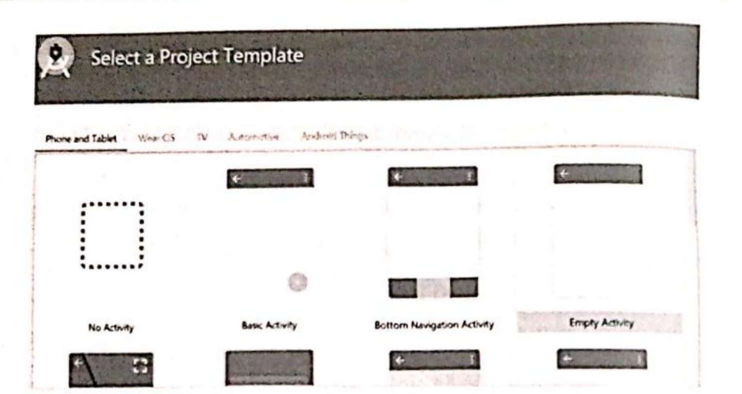
```
public Lugar() {
   fecha = System.currentTimeMillis();
   posicion = new GeoPunto(0.0, 0.0);
   tipo = TipoLugar.OTROS;
}
```


Esto nos permitirá crear un nuevo lugar sin indicar sus atributos.

4. Abre la clase `Principal` y reemplaza el código del método `main()` por:

```
RepositorioLugares lugares = new LugaresLista();
for (int i=0; i<lugares.tamaño(); i++) {
    System.out.println(lugares.elemento(i).toString());
}
```

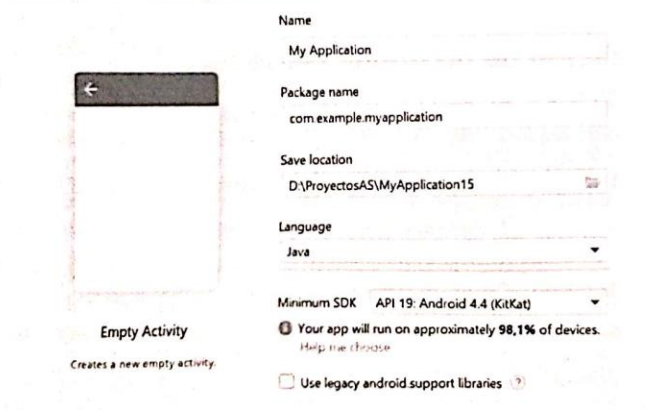
5. Verifica que el resultado es similar al siguiente:



En este curso nos centraremos en las aplicaciones para teléfonos y tabletas, por lo que has de seleccionar siempre la primera pestaña. Has de saber que la plataforma Android también permite desarrollar aplicaciones para dispositivos wearables, Google TV, Android Auto o dispositivos de internet de las cosas.

Observa cómo para este tipo de aplicación puedes elegir diferentes clases de actividades que incorporen ciertos elementos de uso habitual, como menús, botones, anuncios, etc. El concepto de actividad será explicado más adelante. Selecciona *Empty Activity* para añadir una actividad inicial.

3. Pulsa *Next* para pasar a la pantalla donde se rellenan los detalles del proyecto. Puedes dejar los valores por defecto:



A continuación, vemos una descripción para cada campo:

Name: Es el nombre de la aplicación que aparecerá en el dispositivo Android. Tanto en la barra superior, cuando esté en ejecución, como en el icono que se instalará en el menú de programas.

Package name: Indicamos el nombre de paquete de la aplicación. Las clases Java que creemos pertenecerán a este paquete. Como veremos a lo largo del curso, el nombre del paquete también es utilizado por Android para múltiples funciones. Por ejemplo, para determinar en qué directorio se instala la aplicación.

Nota sobre Java/Kotlin: *El nombre del paquete debe ser único en todos los paquetes instalados en un sistema. Por ello, cuando quieras distribuir una aplicación, es muy importante utilizar un dominio que no puedan estar utilizando otras empresas (por ejemplo:* es.upv.elgranlibroandroid.proyecto1*). El espacio de nombres "com.example" está reservado para la documentación de ejemplos (como en este libro) y nunca puede ser utilizado para distribuir aplicaciones. De hecho, Google Play no permite publicar una aplicación si su paquete comienza por "com.example".*

Save location: Permite configurar la carpeta donde se almacenarán todos los ficheros del proyecto.

Language: Selecciona Java o Kotlin, según el lenguaje con el que quieres programar la aplicación.

Minimum API level: Este valor especifica el mínimo nivel de la API que requiere tu aplicación. Por lo tanto, la aplicación no podrá ser instalada en dispositivos con una versión inferior. Procura escoger valores pequeños para que tu aplicación pueda instalarse en la mayoría de los dispositivos. Un valor adecuado puede ser el nivel de API 16 (v4.1), dado que cubriría prácticamente el 100 % de los dispositivos. O el nivel de API 19 (v4.4), que cubriría más del 95 % de los dispositivos. Escoger valores pequeños para este parámetro tiene un inconveniente: no podremos utilizar ninguna de las mejoras que aparezcan en los siguientes niveles de API. Por ejemplo, si queremos utilizar el motor de animaciones de propiedades en nuestra aplicación, tendremos que indicar en este campo la versión 3.0, dado que esta API no aparece hasta esta versión. Pero, en este caso, nuestra aplicación no se podrá instalar en la versión 2.3.

Como se acaba de indicar escoger la versión mínima del SDK es un aspecto clave a la hora de crear un proyecto. Para ayudarnos a tomar esta decisión se indica en negrita el porcentaje de dispositivo donde se podrá instalar nuestra aplicación. En el apartado anterior se explica cómo se obtiene esta información. Pulsa en *Help Me chosee* para visualizar una gráfica donde se muestra los diferentes niveles de API y el porcentaje de usuarios que podrán instalarla la aplicación. Además, si pulsas sobre un nivel te mostrará un resumen con las nuevas características introducidas en este nivel.

4. Deja el resto de los valores por defecto y pulsa *Finish* para crear el proyecto. Deberías tener visible el explorador del proyecto (*Project*) a la izquierda. Abre el fichero *MainActivity* (situado en *app / java / com.example.myapplication*). Debe tener este aspecto:

CAPÍTULO 2.

Diseño de la interfaz de usuario: vistas y *layouts*

El diseño de la interfaz de usuario cobra cada día más importancia en el desarrollo de una aplicación. La calidad de la interfaz de usuario puede ser uno de los factores que conduzca al éxito o al fracaso de todo el proyecto.

Si has realizado alguna aplicación utilizando otras plataformas, advertirás que el diseño de la interfaz de usuario en Android sigue una filosofía muy diferente. En Android la interfaz de usuario no se diseña en código, si no utilizando un lenguaje de marcado en XML similar al HTML.

A lo largo de este capítulo mostraremos una serie de ejemplos que te permitirán entender el diseño de la interfaz de usuario en Android. Aunque no será la forma habitual de trabajar, comenzaremos creando la interfaz de usuario mediante código. De esta forma comprobaremos que cada uno de los elementos de la interfaz de usuario (las vistas) realmente son objetos Java. Continuaremos mostrando cómo se define la interfaz de usuario utilizando código XML. Pasaremos luego a ver las herramientas de diseño integradas en Android Studio. Se describirá el uso de *layouts*, que nos permitirá una correcta organización de las vistas, y el uso de recursos alternativos nos permitirá adaptar nuestra interfaz a diferentes circunstancias y tipos de dispositivos.

En este capítulo también comenzaremos creando la aplicación de ejemplo desarrollada a lo largo del curso, Asteroides. Crearemos la actividad principal, donde simplemente mostraremos cuatro botones, con los que se podrán arrancar diferentes actividades A continuación aprenderemos a crear estilos y temas y los aplicaremos a estas actividades. Para terminar el capítulo propondremos varias prácticas para aprender a utilizar diferentes tipos de vistas y *layouts*.



Objetivos:

- Entender cómo se realiza el diseño de la interfaz de usuario en una aplicación Android.
- Aprender a trabajar con vistas y mostrar sus atributos más importantes.
- Enumerar los tipos de *layouts* que nos permitirán organizar las vistas.
- Mostrar cómo se utilizan los recursos alternativos.
- Aprender a crear estilos y temas para personalizar nuestras aplicaciones.
- Mostrar cómo interactuar con las vistas desde el código Java o Kotlin.
- Describir el uso de *layouts* basados en pestañas (*tabs*).

2.1. Creación de una interfaz de usuario por código

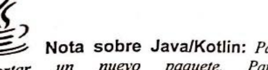
Veamos un primer ejemplo de cómo crear una interfaz de usuario utilizando exclusivamente código Java. Aunque esta no es la forma recomendable de trabajar con Android, resulta interesante para resaltar algunos conceptos.

Ejercicio: *Creación de la interfaz de usuario por código*

1. Abre el proyecto creado en el capítulo anterior y visualiza *MainActivity.java*.
2. Comenta la última sentencia del método `onCreate()` y añade las subrayadas:

```
@Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    //setContentView(R.layout.activity_main);
    TextView texto = new TextView(this);
    texto.setText("Hola, Android");
    setContentView(texto);
}
```

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    //setContentView(R.layout.activity_main)
    val texto = TextView(this)
    texto.text="Hola, Android"
    setContentView(texto)
}
```

Nota sobre Java/Kotlin: *Para poder utilizar el objeto* `TextView` *has de importar un nuevo paquete. Para ello añade al principio* `import android.widget.TextView;`. *Otra alternativa es pulsar **Alt-Intro** para que se añadan automáticamente los paquetes que faltan.*

La interfaz de usuario de Android está basada en una jerarquía de clases descendientes de la clase `View` (vista). Una vista es un objeto que se puede dibujar y se utiliza como un elemento en el diseño de la interfaz de usuario (un botón, una imagen, una etiqueta de texto como la que se ha utilizado en el ejemplo, etc.). Cada uno de estos elementos se define como una subclase de la clase `View`; la subclase para representar un texto es `TextView`.

El ejemplo comienza creando un objeto de la clase `TextView`. El constructor de la clase acepta como parámetro una instancia de la clase `Context` (contexto). Un contexto es un manejador del sistema que proporciona servicios como la resolución de recursos, la obtención de acceso a bases de datos o las preferencias. La clase `Activity` es una subclase de `Context`, y como la clase `MainActivity` es una subclase de `Activity`, también es de tipo `Context`. Por ello, puedes pasar `this` (el objeto actual de la clase `MainActivity`) como contexto del `TextView`.

3. Después se define el texto que se visualizará en el `TextView` mediante `setText()`. Finalmente, mediante `setContentView()` se indica la vista que visualizará la actividad.
4. Ejecuta el proyecto para verificar que funciona.

2.2. Creación de una interfaz de usuario usando XML

En el ejemplo anterior hemos creado la interfaz de usuario directamente en el código. A veces puede ser muy complicado programar interfaces de usuario, ya que pequeños cambios en el diseño pueden corresponder a complicadas modificaciones en el código. Un principio importante en el diseño de *software* es que conviene separar todo lo posible el diseño, los datos y la lógica de la aplicación.

Android proporciona una alternativa para el diseño de interfaces de usuario: los ficheros de diseño basados en XML. Veamos uno de estos ficheros. Para ello accede al fichero *res/layout/activity_main.xml* de nuestro proyecto. Se muestra a continuación. Este *layout* o fichero de diseño proporciona un resultado similar al del ejemplo de diseño por código anterior:

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

NOTA: *Cuando haces doble clic en el explorador del proyecto sobre activity_main.xml, probablemente lo abra en modo gráfico. Para verlo en modo texto, selecciona la pestaña Text.*

Resulta sencillo interpretar su significado. Se introduce un elemento de tipo `CoinstraintLayout`, cuya función, como se estudiará más adelante, es contener otros elementos de tipo `View`. Este `CoinstraintLayout` tiene seis atributos. Los tres primeros, `xmlns:android`, `xmlns:app` y `xmlns:tools`, son declaraciones de espacios de nombres de XML que utilizaremos en este fichero (este tipo de parámetro solo es necesario especificarlo en el primer elemento). Los dos siguientes permiten definir la anchura y altura de la vista. En el ejemplo se ocupará todo el espacio disponible. El último atributo indica la actividad asociada a este *layout*.

Dentro del `CoinstraintLayout` solo tenemos un elemento de tipo `TextView`. Este dispone de varios atributos. Los dos primeros definen el alto y el ancho (se ajustarán al texto contenido). El siguiente indica el texto a mostrar. Los cuatro siguientes indican la posición de la vista dentro del `CoinstraintLayout`.



Ejercicio: *Creación de la interfaz de usuario con XML*

1. Para utilizar el diseño en XML regresa al fichero *MainActivity.java* y deshaz los cambios que hicimos antes (elimina las tres últimas líneas y quita el comentario).
2. Ejecuta la aplicación y verifica el resultado. Ha de ser muy similar al anterior.
3. Modifica el valor de `hello_world` en el fichero *res/values/strings.xml.*
4. Vuelve a ejecutar la aplicación y visualiza el resultado.

Analicemos ahora la línea en la que acabas de quitar el comentario:

```
setContentView(R.layout.activity_main)
```

Aquí, `R.layout.main` corresponde a un objeto `View` que se creará en tiempo de ejecución a partir del recurso *activity_main.xml*. Trabajar de esta forma, en comparación con el diseño basado en código, no quita velocidad y requiere menos memoria. Este identificador es creado automáticamente en la clase `R` del proyecto a partir de los elementos de la carpeta *res*. La definición de la clase `R` puede ser similar a:

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int ic_launcher=0x7f020000;
    }
    public static final class id {
        public static final int action_settings=0x7f070000;
    }
    public static final class layout {
        public static final int activity_main=0x7f030000;
    }
    public static final class menu {
        public static final int main=0x7f060000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    ...
```


NOTA: *Este fichero se genera automáticamente. Nunca debes editarlo.*

Has de tener claro que los identificadores de la clase `R` son meros números que informan al gestor de recursos sobre qué datos ha de cargar. Por lo tanto, no se trata de verdaderos objetos; estos se crearán en tiempo de ejecución solo cuando sea necesario usarlos.



Ejercicio: *El fichero R.java*

1. En **Android Studio,** el fichero R.java no es accesible desde el explorador del proyecto. No obstante, puedes acceder a él si pulsas con el botón derecho sobre *app* y seleccionas *Show in Explorer* (o *Show in Dolfin*). Desde esta carpeta abre el fichero:

   app\build\generated\not_namespaced_r_class_sources\debug\r \nombre\del\paquete\R.java

   Donde nombre\del\paquete has de reemplazarlo por el que corresponda al paquete de tu aplicación.
2. Compáralo con el fichero mostrado previamente. ¿Qué diferencias encuentras? *(RESPUESTA: cambian los valores numéricos en hexadecimal y contiene muchos más identificadores.)*
3. Abre el fichero *MainActivity.java* y reemplaza `R.layout.activity_main` por el valor numérico al que corresponde en *R.java.*
4. Ejecuta de nuevo el proyecto. ¿Funciona? ¿Crees que sería adecuado dejar este valor numérico?
5. Aunque haya funcionado, este valor puede cambiar en un futuro. Por lo tanto, para evitar problemas futuros vuelve a reemplazarlo por `R.layout.activity_main`.



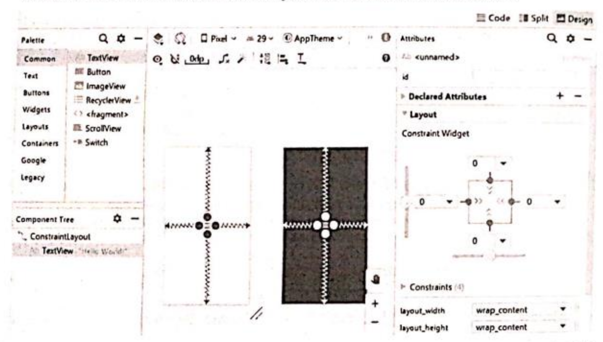
Preguntas de repaso: *Interfaz de usuario en XML y en código*



Preguntas de repaso: *El fichero R.java*

2.2.1. Edición visual de las vistas

Veamos ahora como editar los *layouts* o ficheros de diseño en XML. En el explorador del proyecto abre el fichero *res/layout/activity_main.xml*. Verás que en la parte superior derecha aparecen dos lengüetas: *Code, Split* y *Design*. Con *Code* podrás editar directamente el código XML. Con *Design* podrás realizar este diseño de forma visual. Con *Split* muestra las dos alternativas. Veamos cómo se realizaría el diseño visual. La herramienta de edición de layouts se muestra a continuación:



NOTA: Si aparece un error con problemas de renderizado prueba otros niveles de API, en el desplegable que aparece junto al pequeño robot verde, o con otro tema, en el botón con forma de círculo.

En la parte inferior izquierda encontramos el marco *Component Tree* con una lista con todos los elementos del *layout*. Este *layout* tiene solo dos vistas: un `ConstraintLayout` que contiene un `TextView`. En el marco central aparece una representación de cómo se verá el resultado y a su derecha, con fondo azul, una representación con los nombres de cada vista y su tamaño. En la parte superior aparecen varios controles para representar este *layout* en diferentes configuraciones. Cuando diseñamos una vista en Android, hay que tener en cuenta que desconocemos el dispositivo final donde se visualizará y la configuración específica elegida por el usuario. Por esta razón, resulta importante que verifiques que el *layout* se ve de forma adecuada en cualquier configuración.

En la parte superior, de izquierda a derecha, encontramos los siguientes botones: El primero permite mostrar solo la visualización de diseño, solo la visualización esquemática de vistas o ambas. El botón muestra la orientación horizontal (*landscape*), vertical (*portrait*) y también podemos escoger el tipo de interfaz de usuario (coche, TV, reloj...), con Nexus 4 escogemos el tipo de dispositivo (tamaño y resolución de la pantalla), con 24 la versión de Android, con AppTheme cómo se verá nuestra vista tras aplicar un tema y con Default (en-us) editar las traducciones.

Para editar un elemento, selecciónalo en el marco *Component Tree* o pincha directamente sobre él en la ventana de previsualización. Al seleccionarlo, puedes modificar alguna de sus propiedades en el marco *Properties*, situado a la derecha. Echa un vistazo a las propiedades disponibles para `TextView` y modifica alguna de ellas. En muchos casos te aparecerá un desplegable con las opciones disponibles. Aquí solo se muestra una pequeña parte de las propiedades disponibles. Pulsa en *All properties* para mostrarlas todas.

El marco de la izquierda, *Palette*, te permite insertar de forma rápida nuevas vistas al *layout*. Puedes arrastrar cualquier elemento a la ventana de previsualización o al marco *Component Tree*. En el anexo D se ha incluido una lista con las vistas disponibles.

NOTA: El siguiente vídeo corresponde a una versión anterior de la herramienta. Aunque cambian algunos iconos el funcionamiento continúa siendo similar. Para crear un nuevo layout pulsa con el botón derecho en el explorador de proyecto sobre app y selecciona la opción: New > Android resource file



Vídeo[tutorial]: *Diseño visual de* layouts*: visión general*



Ejercicio: *Creación visual de vistas*

1. Crea un nuevo proyecto con los siguientes datos:

```
Phone and Tablet / Empty Activity
Name: Primeras Vistas
Minimum API level: API 19 Android 4.4 (KitKat)
```

   Deja el resto de los parámetros con los valores por defecto.

2. Abre el fichero *res/layout/activity_main.xml*.
3. Vamos a hacer que la raíz del *layout* se base en un `LinearLayout` vertical. Este tipo de *layout* es uno de los más sencillos de utilizar. Te permite representar las vistas una debajo de la otra. En el marco *Component Tree* pulsa con el botón derecho

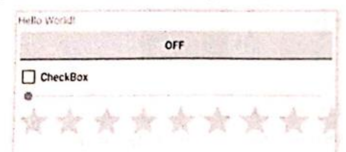
sobre `ConstraintLayout` y selecciona *Covert View...* Indica que quieres usar el layout, `LinearLayout`. El layout que ha añadido es de tipo horizontal. En nuestro caso lo queremos de tipo vertical, para cambiarlo pulsa con el botón derecho sobre `LinearLayout`, y selecciona *LinearLayout/Convert orientation to vertical*.

Todas las operaciones que hacemos en modo diseño visual (lengüeta *Design*) también las podemos hacer con el editor de texto. Para probarlo, deshaz el trabajo anterior, usando la opción Edit / Undo (Ctrl+Z). Selecciona la lengüeta *Code* y cambia la etiqueta `ConstraintLayout` por `LinearLayout`. Añade el atributo `orientation` a `LinearLayout` para que la orientación sea vertical. Elimina los atributos innecesarios del `TextView`, que utiliza con `ConstraintLayout`. El resultado ha de ser similar a:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent" ... />
</LinearLayout>
```

Regresa a la lengüeta *Design*.

4. Desde la paleta de la izquierda arrastra, al área de diseño, los siguientes elementos: `ToggleButton`, `CheckBox`, `SeekBar` y `RatingBar`.



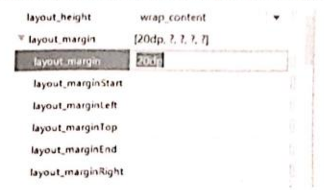
5. Selecciona la primera vista que estaba ya creada (`TextView`) y pulsa *<Supr>* para eliminarla.

6. Selecciona la vista `ToggleButton`. Pulsa ahora (*Set layout width to wrap_content*). Conseguirás que la anchura del botón se ajuste a la anchura de su texto. Pulsa el botón ↔ (*Set layout width to match_parent*) para que el ancho del botón se ajuste a su contenedor. Observa en el marco *Attributes* cómo cambia la propiedad *layout_width*. Si el botón anterior no funciona cámbialo desde este marco. Deja el valor *wrap_content*.

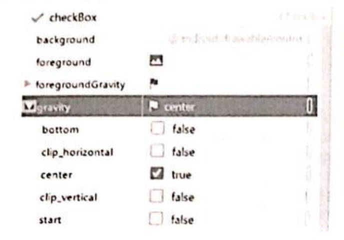
7. Pulsa el botón (*Convert orientation to horizontal*), para conseguir que el `LinearLayout` donde están las diferentes vistas tenga una orientación horizontal. Comprobarás que no caben todos los elementos. Pulsa el botón (*Convert orientation to vertical*), para volver a una orientación vertical.

8. Con la vista `ToggleButton` seleccionada, pulsa el botón (*Set layoutheight to match_parent*). Conseguirás que la altura del botón se ajuste a la altura de su contenedor. El problema es que el resto de los elementos dejan de verse. Vuelve a pulsar este botón para regresar a la configuración anterior (también puedes pulsar *Ctrl-Z*).

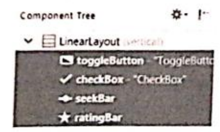
9. Selecciona la vista `CheckBox`. Ve al marco *Attributes* y en la parte inferior pulsa en *All attributos*. Busca la propiedad *layout_margin* en el campo con el mismo nombre introduce "20dp". Se añadirá un margen alrededor de la vista.



10. Busca la propiedad *gravity* y selecciona *center*.



11. Observa que hay un espacio sin usar en la parte inferior del *layout*. Vamos a distribuir este espacio entre las vistas. Desde el marco *Commponet Tree* selecciona las cuatro vistas que has introducido dentro del `LinearLayout`. Para una selección múltiple mantén pulsada la tecla *Ctrl*.



12. Aparecerá un nuevo botón: (*Distribute Weights Evenly*). Púlsalo y la altura de las vistas se ajustará para que ocupen la totalidad del *layout*. Realmente, lo que hace es dividir el espacio sin usar de forma proporcional entre las vistas. Es equivalente a poner `layout_weight` = 1 y `layout_height` = 0dp para todas las vistas de este *layout*. Esta propiedad se modificará en un siguiente punto.

13. Selecciona las cuatro vistas y pulsa el botón (*Clear All Weight*) para eliminar los pesos introducidos.

14. Selecciona la vista `CheckBox`. Asigna, en el marco *Attributes*, `layout_height` = 0dp y `layout_weight` = 1 en esta vista. Observa como toda la altura restante es asignada a la vista seleccionada.



15. Para asignar un peso diferente a cada vista, repite los pasos anteriores donde asignábamos peso 1 a todas las vistas (botón:). Pulsa la lengüeta *Code* y modifica manualmente el atributo `layout_weight` para que el `ToggleButton` tenga valor 2; `CheckBox` tenga valor 0.5; `SeekBar` valor 4 y `RatingBar` valor 1. Pulsa la lengüeta *Design*. Como puedes observar, estos pesos permiten repartir la altura sobrante entre las vistas.

16. Utiliza los siguientes botones: + – 1:1 para ajustar el zum.

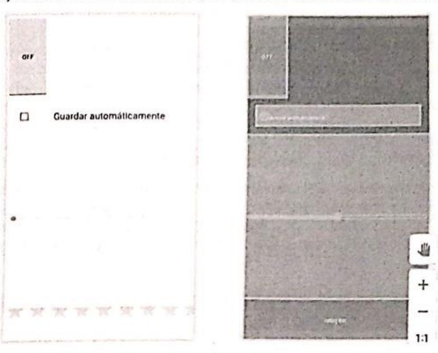
17. Utiliza los botones de la barra superior para observar cómo se representará el *layout* en diferentes situaciones y tipos de dispositivos:

Pixel ⌄ 29 ⌄ AppTheme ⌄ Default (en-us) ⌄

18. Selecciona la vista `CheckBox` y observa las diferentes propiedades que podemos definir en el marco *Attributes*. Algunas ya han sido definidas por medio de la barra de botones. En concreto, y siguiendo el mismo orden que en los botones, hemos modificado: *Layout margin = 20dp, gravity = center* y *Layout weight = 0.5.*

19. Busca la propiedad *Text* y sustituye el valor *CheckBox* por "Guardar automáticamente" y *Text size* por "9pt".

20. Pulsa el botón para mostrar la visualización de diseño, la esquemática (Blueprnt) o ambas. A continuación, se muestra el resultado final obtenido:



21. Pulsa sobre la lengüeta *Code*. A continuación, se muestra este código:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">
    <ToggleButton
        android:id="@+id/toggleButton"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:text="ToggleButton" />
    <CheckBox
        android:id="@+id/checkBox"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_margin="20dp"
        android:layout_weight="0.5"
        android:gravity="center"
        android:text="Guardar automáticament"
        android:textSize="9pt" />
    <SeekBar
        android:id="@+id/seekBar"
        android:layout_width="match_parent"
```

```
        android:layout_height="0dp"
        android:layout_weight="4" />
    <RatingBar
        android:id="@+id/ratingBar"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" />
</LinearLayout>
```

22. Ejecuta el proyecto para ver el resultado en el dispositivo.



Ejercicio: *Vistas de entrada de texto*

1. Añade en la parte superior del *layout* anterior una vista de tipo entrada de texto `EditText`, de tipo normal (*Plain Text*). Lo encontrarás dentro del grupo *Text*. Debajo de esta, una de tipo correo electrónico (*E-mail*) seguida de una de tipo palabra secreta (*Password*). Continua así con otros tipos de entradas de texto.
2. Ejecuta la aplicación.
3. Observa como al introducir el texto de una entrada se muestra un tipo de teclado diferente.

2.2.2. Los atributos de las vistas



Vídeo[**tutorial**]: *Atributos de la clase View en Android*



Vídeo[**tutorial**]: *Atributos de la clase TextView en Android*



Recursos adicionales: *Atributo de dimensión*

En muchas ocasiones tenemos que indicar la anchura o altura de una vista, un margen, el tamaño de un texto o unas coordenadas. Este tipo de atributos se conocen como atributos de dimensión. Dado que nuestra aplicación podrá ejecutarse en una gran variedad de dispositivos con resoluciones muy diversas, Android nos permite indicar estas dimensiones de varias formas. En la siguiente tabla se muestran las diferentes posibilidades:

px (píxeles): Estas dimensiones representan los píxeles en la pantalla.

mm (milímetros): Distancia real medida sobre la pantalla.

in (pulgadas): Distancia real medida sobre la pantalla.

pt (puntos): Equivale a 1/72 pulgadas.

dp (píxeles independientes de la densidad): Presupone un dispositivo de 160 píxeles por pulgada. Si luego el dispositivo tiene otra densidad, se realizará el correspondiente ajuste. A diferencia de otras medidas como mm, in y pt este ajuste se hace de forma aproximada dado que no se utiliza la verdadera densidad gráfica, si no el grupo de densidad en que se ha clasificado el dispositivo (ldpi, mdpi, hdpi...). Esta medida presenta varias ventajas cuando se utilizan recursos gráficos en diferentes densidades. Por esta razón, Google insiste en que se utilice siempre esta medida. Desde un punto de vista práctico un dp equivale aproximadamente a 1/160 pulgadas. Y en dispositivos con densidad gráfica mdpi un dp es siempre un pixel.

sp (píxeles escalados): Similar a dp, pero también se escala en función del tamaño de fuente que el usuario ha escogido en las preferencias. Indicado cuando se trabaja con fuentes.



Recursos adicionales: *Tipos de vista y sus atributos*

Consulta el anexo D para conocer una lista con todos los descendientes de la clase `View` y sus atributos.



Preguntas de repaso: *Las vistas y sus atributos*

2.3. *Layouts*

Si queremos combinar varios elementos de tipo vista, tendremos que utilizar un objeto de tipo *layout*. Un *layout* es un contenedor de una o más vistas y controla su comportamiento y posición. Hay que destacar que un *layout* puede contener otro *layout* y que es un descendiente de la clase View.

La siguiente lista describe los *layout* más utilizados en Android:

LinearLayout: Dispone los elementos en una fila o en una columna.

TableLayout: Distribuye los elementos de forma tabular.

RelativeLayout: Dispone los elementos con relación a otro o al padre.

ConstraintLayout: Versión mejorada de RelativeLayout, que permite una edición visual desde el editor.

FrameLayout: Permite el cambio dinámico de los elementos que contiene.

AbsoluteLayout: Posiciona los elementos de forma absoluta.

Dado que un ejemplo vale más que mil palabras, pasemos a mostrar cada uno de estos *layouts* en acción:

LinearLayout es uno de los *layout* más utilizado en la práctica. Distribuye los elementos uno a continuación de otro, bien de forma horizontal o vertical.

```
<LinearLayout xmlns:android="http://...
    android:layout_height="match_parent"
    android:layout_width="match_parent"
    android: orientation ="vertical">
  <AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
  <CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un checkBox"/>
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"/>
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un texto cualquiera"/>
</LinearLayout>
```



TableLayout distribuye los elementos de forma tabular. Se utiliza la etiqueta <TableRow> cada vez que queremos insertar una nueva línea.

```
<TableLayout xmlns:android="http://...
    android:layout_height="match_parent"
    android:layout_width="match_parent">
  <TableRow>
   <AnalogClock
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"/>
   <CheckBox
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Un checkBox"/>
  </TableRow>
  <TableRow>
   <Button
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Un botón"/>
   <TextView
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="Un texto cualquiera"/>
  </TableRow>
</TableLayout>
```



RelativeLayout permite comenzar a situar los elementos en cualquiera de los cuatro lados del contenedor e ir añadiendo nuevos elementos pegados a estos.

```
<RelativeLayout
    xmlns:android="http://schemas...
    android:layout_height="match_parent"
    android:layout_width="match_parent">
  <AnalogClock
    android:id="@+id/AnalogClock01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentTop="true"/>
  <CheckBox
    android:id="@+id/CheckBox01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/AnalogClock01"
    android:text="Un checkBox"/>
  <Button
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"
    android:layout_below="@id/CheckBox01"/>
  <TextView
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:text="Un texto cualquiera"/>
</RelativeLayout>
```



ConstraintLayout versión más flexible y eficiente de RelativeLayout.

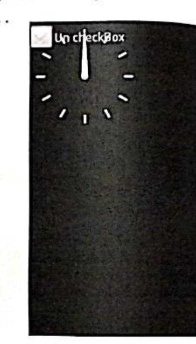
```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas...
    android:layout_height="match_parent"
    android:layout_width="match_parent">
  <AnalogClock
    android:id="@+id/AnalogClock01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toTopOf="parent"/>
  <CheckBox
    android:id="@+id/CheckBox01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un checkBox"
    app:layout_constraintTop_toBottomOf=
                          "@+id/AnalogClock01"
    app:layout_constraintTop_toTopOf="parent"/>
  <Button
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"
```



```
    app:layout_constraintTop_toBottomOf=
                            "@+id/CheckBox01"
    app:layout_constraintLeft_toLeftOf=
                            "@+id/CheckBox01"/>
  <TextView
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un texto cualquiera"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout
```

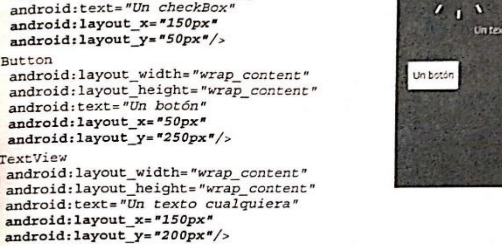
FrameLayout posiciona las vistas usando todo el contenedor, sin distribuirlas espacialmente. Este *layout* suele utilizarse cuando queremos que varias vistas ocupen un mismo lugar. Podemos hacer que solo una sea visible, o superponerlas. Para modificar la visibilidad de un elemento utilizaremos la propiedad *visibility*.

```
<FrameLayout xmlns:android="http://schemas...
    android:layout_height="match_paren"
    android:layout_width="match_parent">
  <AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_conten"/>
  <CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un checkBox"/>
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"
    android:visibility="invisible"/>
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un texto cualquiera"
    android:visibility="invisible"/>
</FrameLayout>
```



AbsoluteLayout permite indicar las coordenadas (*x*, *y*) donde queremos que se visualice cada elemento. No es recomendable utilizar este tipo de *layout*. La aplicación que estamos diseñando tiene que visualizarse correctamente en dispositivos con cualquier tamaño de pantalla. Para conseguirlo, no es una buena idea trabajar con coordenadas absolutas. De hecho, este tipo de *layout* ha sido marcado como obsoleto.


```
<AbsoluteLayout xmlns:android="http://schemas.
    android:layout_height="match_parent"
    android:layout_width="match_parent">
  <AnalogClock
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="50px"
    android:layout_y="50px"/>
  <CheckBox
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un checkBox"
    android:layout_x="150px"
    android:layout_y="50px"/>
  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un botón"
    android:layout_x="50px"
    android:layout_y="250px"/>
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Un texto cualquiera"
    android:layout_x="150px"
    android:layout_y="200px"/>
</AbsoluteLayout>
```



Si tienes dudas sobre cuándo emplear cada layout, usa la siguiente tabla:

LinearLayout: Diseños muy sencillos.

RelativeLayout: Nunca, hay una nueva alternativa.

ConstraintLayout: Usar por defecto.

FrameLayout: Varias vistas superpuestas.

AbsoluteLayout: Nunca. Aunque está bien conocerlo por si acaso.



Vídeo[tutorial]: *Los layouts en Android*



Preguntas de repaso: *Tipos de layouts*



Preguntas de repaso: *Atributos de los layouts*

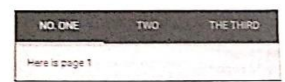
También podemos utilizar otros *layouts*, que se describen a continuación:

ScrollView: Visualiza una vista en su interior; cuando esta no cabe en pantalla, se permite un deslizamiento vertical.

HorizontalScrollView: Visualiza una vista en su interior; cuando esta no cabe en pantalla, se permite un deslizamiento horizontal.



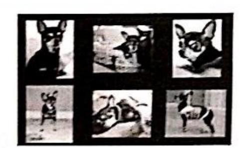
TabLayout, FragmentTabHost, TabLayout o TabHost: Proporciona una lista de pestañas que pueden ser pulsadas por el usuario para seleccionar el contenido a visualizar. Se estudia al final del capítulo.



ListView: Visualiza una lista deslizable verticalmente de varios elementos. Su utilización es algo compleja. Se verá un ejemplo en el capítulo siguiente.



GridView: Visualiza una cuadrícula deslizable de varias filas y varias columnas.



RecyclerView: Versión actualizada que realiza las mismas funciones que `ListView` o `GridView`. Se verá en el siguiente capítulo.

ViewPager: Permite visualizar una serie de páginas, donde el usuario puede navegar arrastrando a derecha o izquierda. Cada página ha de ser almacenada en un fragment.

2.3.1. Uso de ConstraitLayout

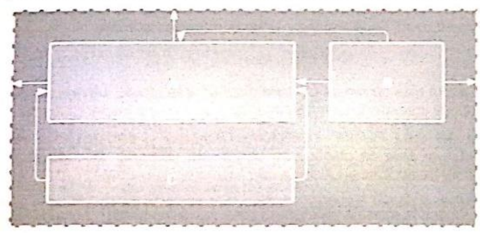


Vídeo[tutorial]: *Uso de ConstraintLayout en Androids*

Este nuevo layout ha sido añadido en una librería de compatibilidad, por lo que se nos anima a usarlo de forma predeterminada. Nos permite crear complejos diseños sin la necesidad de usar *layouts* anidados. El hecho de realizar diseños donde un *layout* se introduce dentro de otro y así repetidas veces, ocasionaba problemas de memoria y eficiencia en dispositivos de pocas prestaciones.

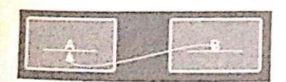
Es muy parecido a *RelativeLayout*, pero más flexible y fácil de usar desde el editor visual de Android Studio (disponible desde la versión 2.3). De hecho, se recomienda crear tu *layout* con las herramientas *drag-and-drop*, en lugar de editar el fichero XML. El resto de *layouts* son más fáciles de crear desde XML.

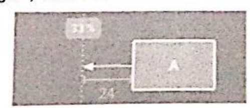
Las posiciones de las diferentes vistas dentro de este *layout* se definen usando *constraint* (en castellano restricciones). Un *constraint* puede definirse en relación al contenedor (*parent*), a otra vista o respecto a una línea de guía (*guideline*). Es necesario definir para cada vista al menos un *constraint* horizontal y uno vertical. No obstante, también podemos definir más de un *constraint* en el mismo eje. Veamos un ejemplo:



Observa cómo la vista A está posicionada con respecto al contenedor. La vista B está posicionada verticalmente con respecto a la vista A, aunque se ha definido un segundo *constraint* con respecto al lado derecho del contenedor. Veremos más adelante cómo se maneja esta circunstancia.

También podemos posicionar las vistas usando alineamientos. Observa cómo el borde superior de B está alineado con el borde superior de A. La vista C define dos alineamientos horizontales simultáneos, pero ninguno vertical, lo que ocasionará un error de compilación. También podemos realizar alineamientos usando la línea base de texto y líneas de guía, como se muestra a continuación:





Los proyectos se encuentran clasificados en categorías: *Admin, Background, Connectivity, Content, Input, Media, Notification, ...* Selecciona un proyecto de alguna de estas categorías. A la derecha podrás leer una breve descripción o ver una vista previa.

2. Pulsa *Next* para pasar a la siguiente ventana. Podrás configurar el nombre de la aplicación, explorar el proyecto accediendo a su sitio web en GitHub e indicar la carpeta donde quieres descargarlo:

Provide information about your project

Application name: Notifications

GitHub URL: https://github.com/android/user-interface-samples/

Project location: D:\ProyectosAS\Notifications

3. Pulsa *Finish* y a continuación ejecuta el proyecto seleccionado.
4. Estudia el código del proyecto.

1.12. Depurar

Programación y errores de código son un binomio inseparable. Por lo tanto, resulta fundamental sacar el máximo provecho a las herramientas de depuración.

1.12.1. Depurar con el entorno de desarrollo

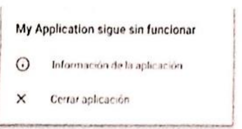
Android Studio integra excelentes herramientas para la depuración de código. Para probarlas, introduce un error en tu código modificando `MainActivity` de forma que en método `onCreate()` tenga este código:

```
@Override
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   Object o = null;
   o.toString();
   setContentView(R.layout.activity_main);
}
```

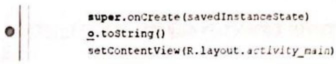
```
lateinit var o: Any

override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   o.toString()
   setContentView(R.layout.activity_main)
}
```

Este cambio introduce **en Java** un `NullPointerException` y **en Kotlin** un `UninitializedPropertyAccessException`. Si ahora ejecutas tu aplicación, te aparecerá algo similar a:



Pulsa *Cerrar* para finalizar la aplicación. Para averiguar más sobre el error, inserta un punto de ruptura (*breakpoint*) en el código fuente en la línea `o.toString()` (el *breakpoint* se introduce haciendo clic en la barra de la izquierda).



```
super.onCreate(savedInstanceState)
o.toString()
setContentView(R.layout.activity_main)
```

Entonces selecciona *Run > Debug 'app'* (Mayús+F9) o pulse en para ejecutarlo en modo *Debug*. Tu aplicación se reiniciará mostrando el siguiente mensaje:

Waiting For Debugger

Application My Application (process com.example.myapplication) is waiting for the debugger to attach

FORCE CLOSE

Pero esta vez quedará suspendida cuando alcance el punto de ruptura que has introducido. Entonces puedes recorrer el código en modo *Debug*, igual que se haría en cualquier otro entorno de programación. Pulsa en *Run > Step Over* (F8) para ir ejecutando las líneas una a una.



Vídeo[tutorial]: *Depurar con Android Studio*

1.12.2. Depurar con mensajes Log

El sistema Android utiliza el fichero *LogCat* para registrar todos los problemas y eventos principales que ocurren en el sistema. Ante cualquier error resulta muy interesante consultarlo para tratar de encontrar su origen.

La clase `Log` proporciona un mecanismo para introducir mensajes desde nuestro código en este fichero. Puede ser muy útil para depurar nuestros programas o para verificar el funcionamiento del código. Disponemos de varios métodos para generar distintos tipos de mensajes:

`Log.e()`: Errors
`Log.w()`: Warnings
`Log.i()`: Information
`Log.d()`: Debugging
`Log.v()`: Verbose

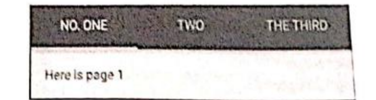
También podemos utilizar otros *layouts*, que se describen a continuación:

ScrollView: Visualiza una vista en su interior; cuando esta no cabe en pantalla, se permite un deslizamiento vertical.

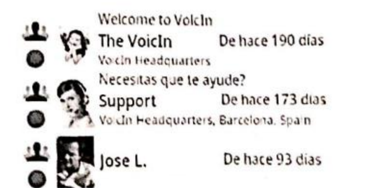
HorizontalScrollView: Visualiza una vista en su interior; cuando esta no cabe en pantalla, se permite un deslizamiento horizontal.



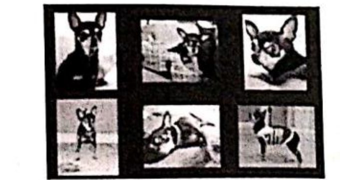
TabLayout, FragmentTabHost, TabLayout o TabHost: Proporciona una lista de pestañas que pueden ser pulsadas por el usuario para seleccionar el contenido a visualizar. Se estudia al final del capítulo.



ListView: Visualiza una lista deslizable verticalmente de varios elementos. Su utilización es algo compleja. Se verá un ejemplo en el capítulo siguiente.



GridView: Visualiza una cuadrícula deslizable de varias filas y varias columnas.



RecyclerView: Versión actualizada que realiza las mismas funciones que `ListView` o `GridView`. Se verá en el siguiente capítulo.

ViewPager: Permite visualizar una serie de páginas, donde el usuario puede navegar arrastrando a derecha o izquierda. Cada página ha de ser almacenada en un fragment.

2.3.1. Uso de ConstraitLayout

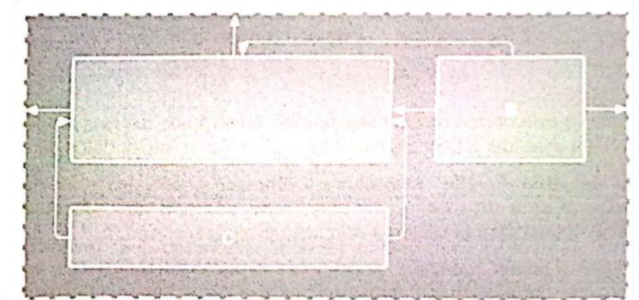


Vídeo[tutorial]: *Uso de ConstraintLayout en Androids*

Este nuevo layout ha sido añadido en una librería de compatibilidad, por lo que se nos anima a usarlo de forma predeterminada. Nos permite crear complejos diseños sin la necesidad de usar *layouts* anidados. El hecho de realizar diseños donde un *layout* se introduce dentro de otro y así repetidas veces, ocasionaba problemas de memoria y eficiencia en dispositivos de pocas prestaciones.

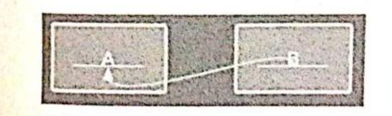
Es muy parecido a *RelativeLayout*, pero más flexible y fácil de usar desde el editor visual de Android Studio (disponible desde la versión 2.3). De hecho, se recomienda crear tu *layout* con las herramientas *drag-and-drop*, en lugar de editar el fichero XML. El resto de *layouts* son más fáciles de crear desde XML.

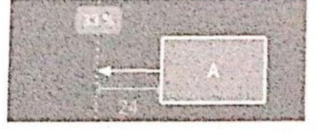
Las posiciones de las diferentes vistas dentro de este *layout* se definen usando *constraint* (en castellano restricciones). Un *constraint* puede definirse en relación al contenedor (*parent*), a otra vista o respecto a una línea de guía (*guideline*). Es necesario definir para cada vista al menos un *constraint* horizontal y uno vertical. No obstante, también podemos definir más de un *constraint* en el mismo eje. Veamos un ejemplo:



Observa cómo la vista A está posicionada con respecto al contenedor. La vista B está posicionada verticalmente con respecto a la vista A, aunque se ha definido un segundo *constraint* con respecto al lado derecho del contenedor. Veremos más adelante cómo se maneja esta circunstancia.

También podemos posicionar las vistas usando alineamientos. Observa cómo el borde superior de B está alineado con el borde superior de A. La vista C define dos alineamientos horizontales simultáneos, pero ninguno vertical, lo que ocasionará un error de compilación. También podemos realizar alineamientos usando la línea base de texto y líneas de guía, como se muestra a continuación:







Ejercicio: *Creación de un layout con ConstraintLayout*

1. Abre el proyecto PrimerasVistas o crea uno nuevo.
2. En Gradle *Scripts/Bulid.gradle (Module:app)* ha de estar la dependencia:

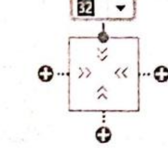
```
dependencies {
  ...
  implementation 'androidx.constraintlayout:constraintlayout:2.0.2'
}
```

3. Abre el *layout activity_main.xml* creado en el ejercicio anterior. Pulsa con el botón derecho en *Component Tree* y selecciona la opción *Convert LinearLayout to ConstraintLayout*. Esta herramienta nos permite convertir nuestros viejos diseños que se basaban en *LinearLayout* o *RelativeLayout* en este nuevo tipo de *layouts*. Por desgracia, no siempre funciona todo lo bien que desearíamos.
4. Crea un nuevo *layout*. Para ello, pulsa con el botón derecho sobre *app/res/layout* y selecciona *New/Layout resource file*. Como nombre introduce "constraint" y en *Root element*: *androidx.constraintlayout.widget.ConstraintLayout*.
5. Si es necesario, desactiva la opción de *Autoconnect* de la barra de acciones del *ConstraintLayou*. Es el segundo icono con forma de imán:

   Tener activa esta opción es útil para diseñar más rápido los *layouts*. No obstante, a la hora de aprender a usar los *constraint*, es mejor ir haciéndolos de uno en uno.
6. Dentro del área *Palette*, selecciona *Common* y arrastra una vista de tipo `ImageView` al área de diseño. Se abrirá una ventana con diferentes recursos *Drawable*. Selecciona la pestaña *Mip Map* y `ic_launcher`.
7. Para definir el primer *constraint*, pulsa sobre el punto de anclaje que aparece en la parte superior del `ImageView` y arrástralo hasta el borde superior del contenedor:



8. En la parte derecha, en la sección *Layout*, nos aparece un editor visual para los *constraint*. Por defecto la distancia seleccionada ha sido 0dp. Pulsa sobre este número y cámbialo a 32dp:



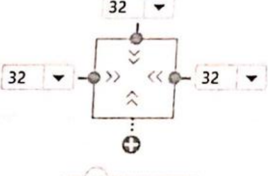
***NOTA:** Según las recomendaciones de Material Design, los márgenes y tamaños han de ser un múltiplo de 8dp.*

9. Realiza la misma operación con el punto de anclaje izquierdo, arrastrándolo al borde izquierdo. Ya tenemos la restricción horizontal y vertical, por lo que la vista está perfectamente ubicada en el *layout*.
10. Arrastra el punto de anclaje derecho al borde derecho. Introduciendo una distancia de 32dp a izquierda y derecha:



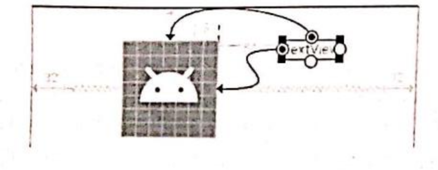
Observa como en este caso, al tener que cumplir simultáneamente dos *constraint* horizontales la imagen se centra horizontalmente. Esto se representa con la línea en zigzag, representando un muelle, que estira de la vista desde los dos lados.

11. Si en lugar de querer la imagen centrada la queremos en otra posición, podemos ir al editor de *constraint* y usar la barra deslizante (*Horizontal Bias*). Desplázala a la posición 25.

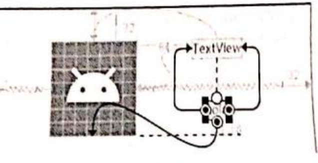


Observa en el área de trabajo como la longitud del muelle de la izquierda es un 25 %, frente al 75 % del muelle de la derecha.

12. Seleccionando el `ImageView` en el área de trabajo, arrastra el cuadrado azul de la esquina inferior derecha hasta aumentar su tamaño a 96x96 dp (múltiplos de 8). Otra alternativa es modificar los valores `layout_width` y `layout_height` en el área *Properties*.
13. Desde el marco *Palette / Common*, añade un `TextView` a la derecha del `ImageView`. Arrastra el punto de anclaje de la izquierda hasta el punto de la derecha de la imagen y establece un margen de 64 dp. Arrastra el punto de anclaje superior del `TextView` hasta el punto superior de la imagen:



14. Añade un nuevo `TextView` bajo el anterior, con texto *"hola"*. Introduce tres *constraint*, usando los puntos de anclaje inferior, izquierdo y derecho, tal y como se muestra en la siguiente figura:



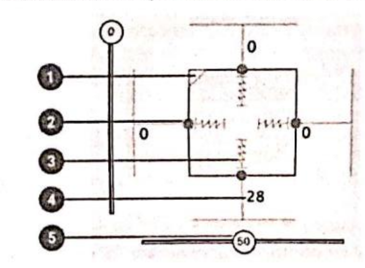
El margen inferior ha de ser 16dp y el izquierdo y derecho 0. De esta forma hemos centrado horizontalmente los dos `TextView`.

15. También podemos conseguir que el ancho del nuevo `TextView` coincida con el superior. Para ello selecciona la vista y en el campo `layout_width` e introduce `match_constraint` o `0dp`. Con esto, hacemos que el ancho se calcule según las restricciones de los *constraint*.



16. Haz que desde `MainActivity` se visualice este *layout* y ejecuta el proyecto en un dispositivo.

Una vez familiarizados con los conceptos básicos de los *constraint* vamos a ver con más detalle las herramientas disponibles. Veamos en editor de *constraint*:



(1) **relación de tamaño:** Puede establecer el tamaño de la vista en una proporción, por ejemplo a 16:9, si al menos una de las dimensiones de la vista está configurada como "ajustar a *constraint*" (0dp). Para activar la relación de tamaño, haz clic donde señala el número 1.

(2) **eliminar constraint:** Se elimina la restricción para este punto de anclaje.

(3) **establecer alto/ancho:** Para cambiar la forma en la que se calcula las dimensiones de la vista, pulsa en este elemento. Existen tres posibilidades:

>>> ajustar a contenido: equivale al valor `warp_content`. (Ej. 1er `TextView`)

ajustar a *constraint*: equivale a poner 0dp. (Ej. 2° `TextView`)

tamaño fijo: equivale a poner un valor concreto de dp. (Ej. `ImageView`) Aunque se representan 4 segmentos, realmente podemos cambiar 2, los horizontales para el ancho y los verticales para el alto.

(4) ***establecer margen:*** Podemos cambiar los márgenes de la vista.

(5) ***sesgo del constraint:*** Ajustamos cómo se reparte la dimensión sobrante.

También es importante repasar las acciones disponibles cuando trabajamos con `ConstraintLayout`:

8dp

Ocultar *constraint*: Elimina las marcas que muestran las restricciones y los márgenes existentes.

Autoconectar: Al añadir una nueva vista se establecen unos *constraint* con elementos cercanos de forma automática.

8dp **Definir márgenes:** Se configura los márgenes por defecto.

Borrar todos los constraint: Se eliminan todas las restricciones del *layout*.

Crear automáticamente constraint: Dada una vista seleccionada, se establecen unos *constraint* con elementos cercanos de forma automática.

Empaquetar / expandir: Se agrupan o se separan los elementos.

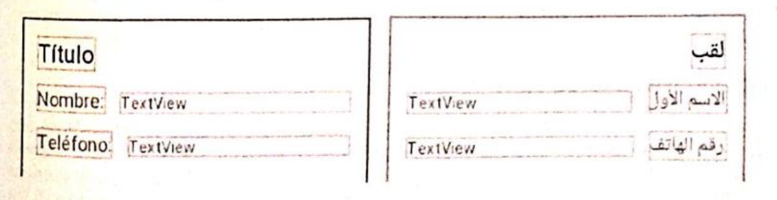
Alinear: Centra o justifica los elementos seleccionados.

Añadir línea de guía: Se crea una nueva línea de referencia.

Uso de escritura derecha a izquierda

Android permite adaptar los layouts a la escritura derecha-izquierda usada en árabe o hebreo. Cuando un dispositivo esté configurado de esta forma, también los formularios se mostrarán de derecha a izquierda.

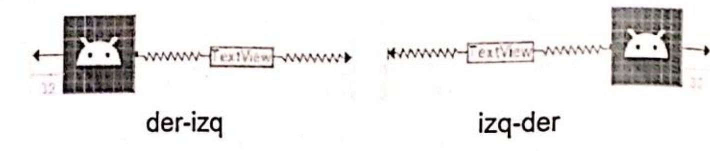
El siguiente ejemplo muestra un formulario y cómo nos interesaría que se viera en un dispositivo configurado en árabe.



Para conseguir este efecto, en lugar de utilizar los atributos tradicionales basados en `Left` y `Right`, usaremos atributos basados en `Start` y `End`. Veamos un ejemplo:

```
<androidx.constraintlayout.widget.ConstraintLayout …>
   <ImageView android:id="@+id/imageView" …
       android:layout_marginStart="32dp"
       app:layout_constraintStart_toStartOf="parent"/>
   <TextView …
       app:layout_constraintStart_toEndOf="@+id/imageView"
       app:layout_constraintEnd_toEndOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Dependiendo de la configuración del dispositivo se mostrará de la siguiente forma:



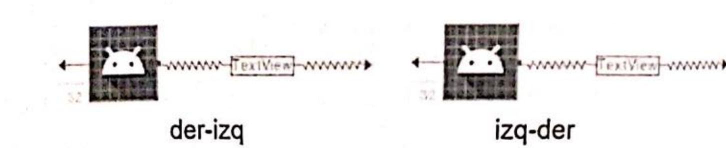
der-izq izq-der

Como puedes observar, en el primer caso `Start` y `End`, significan `Left` y `Right`. Y en el segundo, significan `Right` y `Left`.

Si quieres que el layout se vea igual en ambos casos, utiliza atributos basados en `Left` y `Right`.

```
<androidx.constraintlayout.widget.ConstraintLayout …>
   <ImageView android:id="@+id/imageView" …
       android:layout_marginLeft="32dp"
       app:layout_constraintLeft_toLeftOf="parent"/>
   <TextView …
       app:layout_constraintLeft_toRightOf="@+id/imageView"
       app:layout_constraintRight_toRightOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

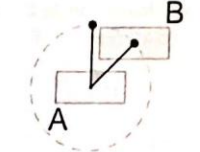
El resultado será:



der-izq izq-der

Posicionamiento Circular

Podemos situar una vista respecto a otra usando coordenadas polares. Para ello tendremos que indicar tres atributos: la vista de donde partimos, el radio (o distancia entre los centros de ambas vistas) y el ángulo (en grados de 0° en la parte superior a 360°). En el ejemplo la vista B se sitúa respecto a la A, con una distancia de 100dp y 45°.




```
<ImageView
    …
    android:id="@+id/B"
    app:layout_constraintCircle="@+id/A"
    app:layout_constraintCircleAngle="45"
    app:layout_constraintCircleRadius="100dp"/>
```

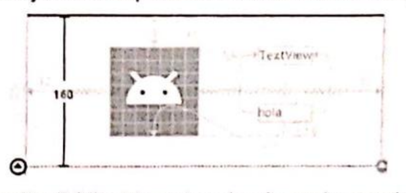


Vídeo[tutorial]: *Cadenas y Líneas Guía en ConstraintLayout*



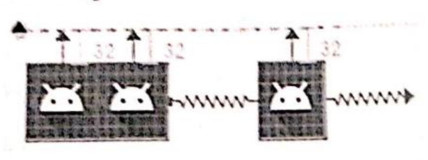
Ejercicio: *Líneas guía y cadenas en ConstraintLayout*

1. Siguiendo con el *layout* del ejercicio anterior. Pulsa en la acción *Guideline* y selecciona *Add Horizontal GuideLine*. Aparecerá un círculo gris con un pequeño triángulo pegado al borde izquierdo, desde donde sale una línea guía horizontal. Arrástrala hacia abajo hasta separarla una distancia de 160dp.

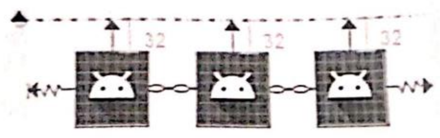


2. Esta guía nos permite dividir el *layout* en dos áreas: la superior, donde ya hemos realizado una especie de cabecera, y la inferior. A partir de ahora los elementos de la parte inferior los colocaremos en relación a esta línea guía.
3. Selecciona el `ImageView`, cópialo (Ctrl+C) y pégalo tres veces (Ctrl+V). Acabamos de hacer tres copias de la imagen, pero no son visibles al estar en la misma posición. En el área *Component Tree*, selecciona `imageView2` y arrástralo bajo la línea guía, pegado a la izquierda. Coloca `imageView3` bajo la línea guía, en el centro, e `imageView4`, a la derecha de este.
4. Selecciona `imageView2` con el botón derecho y borra todos sus *constraint* seleccionando *Clear Constraint of Selección*. Repite esta operación para `imageView3` e `imageView4`.
5. Las tres imágenes han de tener un *constraint* desde el punto de anclaje superior a la línea guía con un margen de 32dp. Desde el punto de anclaje izquierdo de

imageView2, establece un *constraint* con el borde izquierdo. En las otras dos imágenes, del izquierdo al punto de anclaje derecho de la vista de su izquierda. El punto de anclaje derecho de la tercera vista únelo al borde derecho. El resultado ha de ser el siguiente:



6. Para conseguir que estas tres vistas formen una cadena, selecciónalas y utiliza la acción *Align* / *Horizontaly* o el botón derecho *Chains* / *Create Horizontal Chain*:



Observa cómo las vistas ahora están unidas por medio de un conector de cadena. Si abres la lengüeta *Code* para estudiar el XML, puedes comprobar que para establecer la cadena se han añadido dos *constraint*, desde el punto de anclaje derecho de las dos primeras vistas hacia la vista de su izquierda. Es decir, una restricción mutua: de A -> B y de B -> A.

NOTA: *En la versión actual del editor visual, parece que establecer la cadena indicando estos constraint individualmente no parece posible. Hay que usar la acción Align / Horizontaly o Chains / Create Horizontal Chain.*

7. También es posible otras distribuciones de cadena. Podemos hacer que los márgenes se distribuyan solo entre las vistas o solo en los extremos izquierdo y derecho:



Si abres la lengüeta *Code* puedes comprobar que estas dos nuevas configuraciones de cadena se consiguen añadiendo en la primera vista el atributo:

```
app:layout_constraintHorizontal_chainStyle="spread_inside"
```

Para la primera distribución y para la segunda:

```
app:layout_constraintHorizontal_chainStyle="packed"
```

Para la distribución del punto anterior el valor es "spread" o no indicar nada.

8. Existe otra distribución en la que los márgenes desaparecen y se ajusta el ancho de las vistas hasta cubrir todo el espacio disponible. Selecciona la vista central y

en el editor de *constraint* pulsa sobre el icono hasta que aparezca. Recuerda que esta acción es equivalente a poner 0 en layout_width. El resultado se muestra a la izquierda:



Para conseguir el resultado de la derecha, hemos repetido la operación con las otras dos imágenes.

Si en lugar de repartir los anchos por igual, quieres otra configuración, puedes usar el atributo layout_constraintHorizontal_weight (funciona igual que layout_weight en un LinearLayout).

9. *Ejecuta el proyecto en un dispositivo.*

Al crear constraint, recuerde las siguientes reglas:

- *Cada vista debe tener al menos una restricción horizontal y otra vertical.*
- *Solo puede crear restricciones que compartan el mismo plano. Así, el plano vertical (los lados izquierdo y derecho) de una vista puede anclarse solo a otro plano vertical.*
- *De cada punto de restricción (superior, inferior, derecha o izquierda) solo puede salir una flecha. Pero pueden llegar varias flechas desde diferentes vistas.*



Práctica: *Uso de layouts*

1. Utiliza un ConstraintLayout para realizar un diseño similar al siguiente:

Escribe tu nombre
Nombre
CANCELAR ACEPTAR

2. Utiliza un TableLayout para realizar un diseño similar al siguiente:



3. Utiliza un LinearLayout horizontal que contenga en su interior otros LinearLayout para realizar un diseño similar al siguiente. Ha de ocupar toda la pantalla:



4. Visualiza el resultado obtenido en diferentes tamaños de pantalla. ¿Se visualiza correctamente?
5. Realiza el ejercicio de la calculadora usando un `ConstraintLayout`.



Preguntas de repaso: *ConstraintLayout*

2.4. Una aplicación de ejemplo: Asteroides

A lo largo de este libro vamos a ir creando una aplicación de ejemplo que toque los aspectos más significativos de Android. Comenzamos en este capítulo creando una serie de vistas que nos permitirán diseñar una sencilla interfaz de usuario. Si quieres ver cómo quedará la aplicación una vez termines el libro, puedes ver el siguiente vídeo:



Vídeo[tutorial]: *Asteroides*



Enlaces de interés:

- *Asteroides*: Puedes descargarte la aplicación de Google Play:

https://play.google.com/store/apps/details?id=es.upv.mmoviles.asteroides&hl



Práctica: *Creación de la aplicación Asteroides*

1. Crea un nuevo proyecto con los siguientes datos:

```
Phone and Tablet / Empty Activity
Name: Asteroides
Package name: org.example.asteroides
Language: Java
Minimum API level: API 19 Android 4.4 (KitKat)
```

***NOTA:** Deja el resto de los parámetros con su valor por defecto.*

2. Abre el fichero *res/layout/* activity_*main.xml* y trata de crear una vista similar a la que ves a continuación. Ha de estar formada por un `LinearLayout` que contiene un `TextView` y cuatro `Button`. Trata de utilizar recursos para introducir los cinco textos que aparecen.





Solución:

1. El fichero *activity_main.xml* ha de ser similar al siguiente:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:gravity="center"
   android:padding="30dp"
   tools:context=".MainActivity" >
 <TextView android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:text="@string/tituloAplicacion"
     android:gravity="center"
     android:textSize="25sp"
     android:layout_marginBottom="20dp"/>
 <Button android:id="@+id/button01"
     android:layout_height="wrap_content"
     android:layout_width="match_parent"
     android:text="@string/Arrancar"/>
 <Button android:id="@+id/button02"
     android:layout_height="wrap_content"
     android:layout_width="match_parent"
     android:text="@string/Configurar"/>
 <Button android:id="@+id/button03"
     android:layout_height="wrap_content"
     android:layout_width="match_parent"
     android:text="@string/Acercade"/>
 <Button android:id="@+id/button04"
     android:layout_height="wrap_content"
     android:layout_width="match_parent"
```

```
        android:text="@string/Salir"/>
</LinearLayout>
```

2. El fichero *res/values/strings.xml* ha de tener el siguiente contenido:

```
<resources>
    <string name="Arrancar">Jugar</string>
    <string name="Configurar">Configurar</string>
    <string name="Acercade">Acerca de </string>
    <string name="Salir">Salir</string>
    <string name="tituloAplicacion">Asteroides</string>
    <string name="app_name">Asteroides</string>
    <string name="action_settings">Configuración</string>
</resources>
```



Práctica: *Uso de ConstraintLayout en Asteroides*

Repite la práctica anterior pero esta vez usando un `ConstraintLayout`:

2.5. La aplicación Mis Lugares

En este libro vamos a crear una segunda aplicación con características muy diferentes de Asteroides. Tendrá por nombre Mis Lugares y permitirá que los usuarios guarden información relevante sobre los sitios que suelen visitar (restaurantes, tiendas, etc.). En el capítulo 1 se implementaron algunas clases de esta aplicación y se presentó un vídeo que describía su funcionamiento. A diferencia de Asteroides, esta aplicación utilizará un diseño y varios elementos de Material Design. Una explicación más extensa de estos conceptos se abordará en *El Gran Libro de Android Avanzado*. Para más información sobre Material Design y el siguiente ejercicio te recomendamos un tutorial [17].



Vídeo[tutorial]: *Mis Lugares*

Ejercicio: *Creación de la aplicación Mis Lugares*

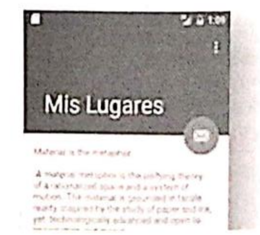
1. Crea un nuevo proyecto con los siguientes datos:

```
Phone and Tablet / Scrolling Activity
Name: Mis Lugares
Package name: com.example.mislugares
Language: Java ó Kotlin
Minimum API level: API 19 Android 4.4 (KitKat)
```

[17] http://www.androidcurso.com/index.php/688

De esta forma, se creará una aplicación con una actividad basada en Material Design. Dispondrá de una barra de acciones y un botón flotante. El contenido principal podrá desplazarse, a la vez que la barra de acciones cambia de tamaño.

2. En el navegador de proyecto pulsa con el botón derecho sobre la clase `ScrollingActivity` y selecciona Refactor > Rename. Introduce como nuevo nombre `MainActivity`. En la carpeta *res/layout* renombra el fichero *activity_scrolling.xml* por *activity_main.xml*. En la misma carpeta renombra *content_scrolling.xml* por *content_main.xml*. Finalmente, en la carpeta *res/menu* renombra *menu_scrolling.xml* por *menu_main.xml*.
3. Ejecuta el proyecto y verifica su comportamiento:



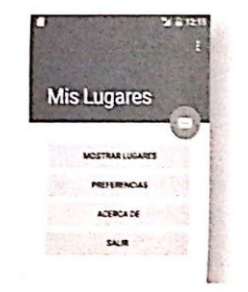
NOTA: Para profundizar en Material Design puedes hacer los siguientes tutoriales [18].



Práctica: *Creación de una primera actividad en Mis Lugares*

En esta práctica crearemos una primera actividad que contendrá simplemente cuatro botones.

1. Edita el fichero *res / layout / content_main.xml* y trata de crear una vista similar a la que ves a continuación. Has de dejar el `NestedScrollView` que ya tenías y reemplazar el `TextView` por un `ConstraintLayout` o `LinearLayout` que contenga cuatro `Button`. Un `NestedScrollView` solo puede contener dentro un elemento, por lo que no puedes introducir directamente los cuatro botones.



[18] http://www.androidcurso.com/index.php/688 http://www.androidcurso.com/index.php/689



Solución: http://www.androidcurso.com/index.php/115



Práctica: *Un formulario para introducir nuevos lugares*

El objetivo de esta práctica es crear un *layout* que permita introducir y editar lugares en la aplicación Mis Lugares.

1. Crea un nuevo *layout* con nombre *edicion_lugar.xml*.
2. Ha de parecerse al siguiente formulario. Puedes basarte en un `LinearLayout` o un `ConstraintLayout` para distribuir los elementos. Pero es importante que este *layout*, se encuentre dentro de un `NestedScrollView` para que cuando el formulario no quepa en pantalla se pueda desplazar verticalmente.

Nombre:
algo que identifique el lugar
Tipo:
Direccion:
direccion del lugar
Telefono: telefono para contactar
Url:
pagina web
Comentario:
introduce tus notas

3. Introduce a la derecha del `TextView` con texto "Tipo:" un `Spinner` con *id* `tipo`. Más adelante configuraremos esta vista para que muestre un desplegable con los tipos de lugares.
4. Las vistas `EditText` han de definir el atributo *id* con los valores: `nombre`, `direccion`, `telefono`, `url` y `comentario`. Utiliza también el atributo `hint` para dar indicaciones sobre el valor a introducir. Utiliza el atributo `inputType` para indicar qué tipo de entrada esperamos. De esta manera se mostrará un teclado adecuado (por ejemplo, si introducimos un correo electrónico aparecerá la tecla @).

***NOTA:** El atributo `inputType` admite los siguientes valores (en negrita los que has de utilizar en este ejercicio): none, text, textCapCharacters, textCapWords, textCapSentences, textAutoCorrect, textAutoComplete, **textMultiLine**, textImeMultiLine, textNoSuggestions, **textUri**, textEmailAddress, textEmailSubject, textShortMessage, textLongMessage, textPersonName, **textPostalAddress**, textPassword, textVisiblePassword, textWebEditText, textFilter, textPhonetic, textWebEmailAddress, textWebPassword, number, numberSigned, numberDecimal, numberPassword, **phone**, datetime, date y time.*

5. Abre la clase `MainActivity` y en el método `onCreate()` reemplaza el layout:

```
setContentView(R.layout.activity_main.edicion_lugar)
```

6. Comenta todas las líneas de este método que hay debajo usando /* ... */. Como ya no se crea el layout *activity_main* los id de vista a los que se accede ya no existen.
7. Ejecuta la aplicación y verifica cómo cambia el tipo de teclado en cada `EditText`.
8. Deshaz el cambio realizado en el punto 5 y 6.



Solución: http://www.androidcurso.com/index.php/115

2.6. Recursos alternativos

Una aplicación Android va a poder ser ejecutada en una gran variedad de dispositivos. El tamaño de pantalla, la resolución o el tipo de entradas puede variar mucho de un dispositivo a otro. Por otra parte, nuestra aplicación ha de estar preparada para diferentes modos de funcionamiento, como el modo "automóvil" o el modo "noche", y para poder ejecutarse en diferentes idiomas.

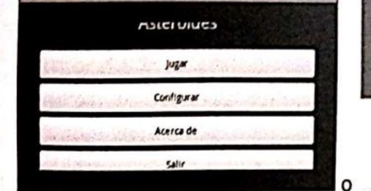
A la hora de crear la interfaz de usuario, hemos de tener en cuenta todas estas circunstancias. Afortunadamente, la plataforma Android nos proporciona una herramienta de gran potencia para resolver este problema: el uso de los recursos alternativos.

***NOTA:** Las prácticas de este apartado se proponen para la aplicación Asteroides, pero también pueden realizarse con Mis Lugares.*

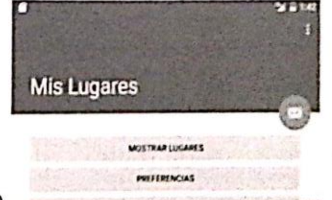


Práctica: *Recursos alternativos en Asteroides*

1. Ejecuta la aplicación Asteroides (o Mis Lugares).
2. Los teléfonos móviles basados en Android permiten cambiar la configuración en apaisado y en vertical. Para conseguir este efecto con el emulador, pulsa el botón ⟲. Si usas un dispositivo de pantalla pequeña, observas como el resultado de la vista que acabas de diseñar en vertical no queda todo lo bien que desearíamos.



o

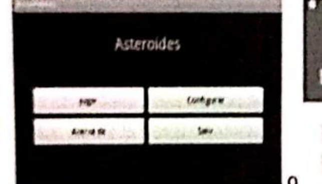


Para resolver este problema, Android te permite diseñar una vista diferente para la configuración horizontal y otra para la vertical.

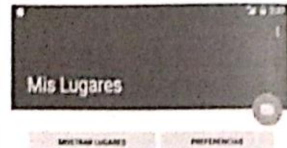
3. Pulsa con el botón derecho sobre la carpeta *res/layout* y selecciona *New > Layout resource file*. Aparecerá una ventana donde has de rellenar en *File name*: activity_main (content_main en Mis Lugares). En *Available qualifiers*: selecciona *Orientation* y pulsa en el botón >>. En el desplegable *Screen orientation*: selecciona *Landscape*. Pulsa en *OK*. Observa como ahora hay dos recursos para el fichero *activity_main.xml* (content_main en Mis Lugares). El primero es el recurso por defecto, mientras que el segundo es el que se usará cuando el dispositivo esté en orientación *Landscape*.



4. Crea en el nuevo layout (*land*) una vista similar a la que ves a continuación: formada por un `TableLayout` con dos `Button` por columna.



o



5. Ejecuta de nuevo la aplicación y observa como la vista se ve correctamente en las dos orientaciones.

Solución:

Has de obtener un código XML similar al siguiente:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:padding="30dp"
    tools:context=".MainActivity" >
  <TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/app_name"
    android:gravity="center"
    android:textSize="25sp"
    android:layout_marginBottom="20dp"/>
  <TableLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:stretchColumns="*">
    <TableRow>
      <Button android:id="@+id/button01"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/accion_mostrar"/>
      <Button android:id="@+id/button02"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/accion_preferencias"/>
    </TableRow>
    <TableRow>
      <Button android:id="@+id/button03"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/accion_acerca_de"/>
      <Button android:id="@+id/button04"
        android:layout_height="wrap_content"
        android:layout_width="match_parent"
        android:text="@string/accion_salir"/>
    </TableRow>
  </TableLayout>
</LinearLayout>
```


NOTA: *Para conseguir que en un* `TableLayout` *las columnas se ajusten a todo el ancho de la tabla, poner* `stretchColumns="*"`. `stretchColumns="0"` *significa que se asigne la anchura sobrante a la primera columna.* `stretchColumns="1"` *significa que se asigne la anchura sobrante a la segunda columna.* `stretchColumns="*"` *significa que se asigne la anchura sobrante entre todas las columnas.*

Android utiliza una lista de sufijos para expresar recursos alternativos. Estos sufijos pueden hacer referencia a la orientación del dispositivo, el lenguaje, la región, la densidad de píxeles, la resolución, el método de entrada, etc.

Por ejemplo, si queremos traducir nuestra aplicación al inglés, español y francés, siendo el primer idioma el usado por defecto, crearíamos tres versiones del fichero *strings.xml* y lo guardaríamos en los tres directorios siguientes:

```
res/values/strings.xml
res/values-es/strings.xml
res/values-fr/strings.xml
```

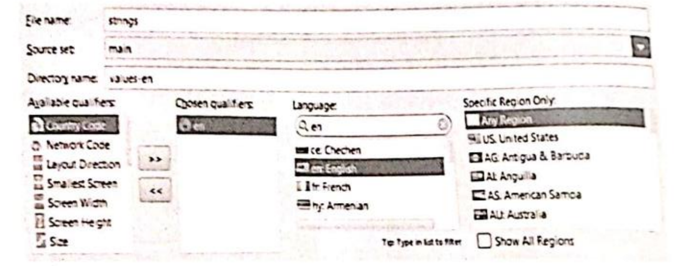
Aunque internamente el SDK de Android utiliza la estructura de carpetas anterior, en Android Studio el explorador del proyecto muestra los recursos alternativos de la siguiente manera:

```
res
  values
    strings.xml(3)
      strings.xml
      strings.xml(es)
      strings.xml(fr)
```



Ejercicio: *Traducción de la aplicación*

1. En Asteroides o Mis Lugares, crea un nuevo recurso alternativo para *strings.xml (en)*: Pulsa con el botón derecho en *res/values*, selecciona *New > Values resource file* e introduce *strings* como nombre de fichero. Mueve el cualificador *Locale* (o *Language*) desde el marco de la izquierda a la derecha, pulsando el botón >>. En *Language* selecciona *English*, en *Specific Region Only* deja el valor *Any Region* y pulsa OK.



NOTA: Observa cómo además del idioma también permite seleccionar la región. De esta forma podremos diferenciar entre inglés americano, británico, australiano, ...

2. Copia el contenido del recurso por defecto, *strings.xml*, al recurso para inglés, *strings.xml (en)*. Traduce los textos al inglés. No has de traducir los nombres de los identificadores de recursos (`accion_mostrar`, `app_name`, ...) estos han de ser igual en todos los idiomas.
3. Ejecuta la aplicación.
4. Vamos a cambiar la configuración de idioma en un dispositivo Android. Para ello accede a *Ajustes* del dispositivo (*Settings*) y selecciona la opción *Personal > Idioma y entrada*. Dentro de esta opción selecciona como idioma *Español* o *Inglés*. *NOTA: Observa que en otros idiomas permite seleccionar tanto el idioma como la región. Por desgracia, para el español solo permite dos regiones: España y Estados Unidos.*
5. Observa como el texto aparece traducido al idioma seleccionado.

Otro ejemplo de utilización de recursos diferenciados lo podemos ver con el icono que se utiliza para lanzar la aplicación. Observa cómo, al crear una aplicación, este icono se crea en cinco carpetas *mipmap* diferentes, para utilizar un icono distinto según la densidad de píxeles del dispositivo:

```
res/mipmap-mdpi/ic_launcher.png
res/mipmap-hdpi/ic_launcher.png
res/mipmap-xhdpi/ic_launcher.png
res/mipmap-xxhdpi/ic_launcher.png
res/mipmap-xxxhdpi/ic_launcher.png
```

NOTA: En el siguiente capítulo se describe por qué se actúa de esta manera.

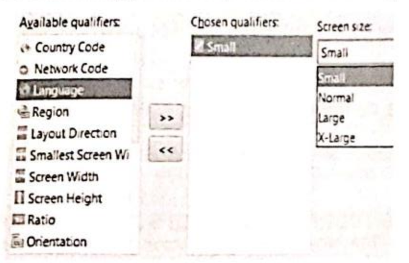
Resulta posible indicar varios sufijos concatenados; por ejemplo:

```
res/values-en-rUS/strings.xml
res/values-en-rUK/strings.xml
```

Pero cuidado, Android establece un orden a la hora de encadenar sufijos. Puedes encontrar una lista de estos sufijos en el apéndice C y en este enlace:

http://developer.android.com/guide/topics/resources/providing-resources.html

Para ver los sufijos disponibles, también puedes pulsar con el botón derecho sobre una carpeta de recursos y seleccionar *New > Android resource file*. Esta opción te permite crear un nuevo recurso y poner el sufijo deseado de forma y orden correctos.



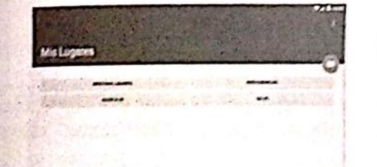


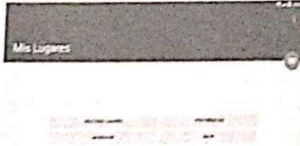
Vídeo[tutorial]: *Uso de recursos alternativos en Android*



Ejercicio: *Creando un* layout *para tabletas*

Si ejecutas la aplicación Asteroides (o Mis Lugares) en una tableta, observarás que los botones son demasiado alargados (izquierda). Queremos que en este caso la apariencia sea similar a la mostrada a la derecha:





1. Crea un recurso alternativo a res/values/dimens.xml, que sea utilizado en pantallas de tamaño `xlarge` (7-10,5 pulgadas) en orientación `land` (apaisado). En este fichero define el siguiente valor:

```
<resources>
    <dimen name="margen_botones">150dp</dimen>
</resources>
```

2. Añade el mismo valor al recurso por defecto, pero esta vez con `30dp`.
3. Modifica los ficheros *res/layout/content_main.xml* y *content_main.xml* (... reemplazando `android:padding="30dp"` por `android:padding="@di... margen_botones"`.
4. Verifica que la aplicación se visualiza correctamente en todos los tipos ... pantalla, tanto en horizontal como en vertical.



Preguntas de repaso: *Recursos alternativos*



Enlaces de interés: *Recursos alternativos*

http://www.androidcurso.com/images/dcomg/ficheros/recursos_alternativos.p...

2.7. Tipos de recursos y recursos del sistema

La definición de los recursos en Android es un aspecto muy importante en el dise... de una aplicación. Una de sus principales ventajas es que facilita a los diseñadores gráficos e introductores de contenido trabajar en paralelo con los programadores.

Añadir un recurso a nuestra aplicación es muy sencillo, no tenemos más que añadir un fichero dentro de una carpeta determinada de nuestro proyecto. Para cada uno de los recursos que añadamos el sistema crea, de forma automática, un id de recurso dentro de la clase `R`.

2.7.1. Tipos de recursos

Según la carpeta que utilicemos, el recurso creado será de un tipo específico. Pasamos a enumerar las carpetas y los tipos posibles:

Carpeta `identificador`	**Descripción**
res/drawable/ `R.drawable`	Ficheros en *bitmap* (*.png*, *.jpg* o *.gif*). Ficheros PNG en formato Nine-patch (*.9.png*). Ficheros XML con descriptores gráficos (véase clase Drawable).
res/mipmap/ `R.mipmap`	Ficheros en *bitmap* (*.png*, *.jpg* o *.gif*). Estos gráficos no son rescaldados para adaptarlos a la densidad gráfica del dispositivo, si no que se buscará en las subcarpetas el gráfico con la densidad más parecida y se utilizará directamente.
res/layout/ `R.layout`	Ficheros XML con los *layouts* usados en la aplicación.
res/menu/ `R.menu`	Ficheros XML con la definición de menús, que podemos asignar a una actividad o a una vista.
res/anim/ `R.anim`	Ficheros XML que permiten definir animaciones Tween, también conocidas como animaciones de vista.
res/animator `R.animator`	Ficheros XML que permiten modificar las propiedades de un objeto a lo largo del tiempo (véase apartado "Animación de propiedades"). Solo desde la versión 3.0.
res/xml/ `R.xml`	Otros ficheros XML, como los ficheros de preferencias.
res/raw/ `R.raw`	Ficheros que se encuentran en formato binario. Por ejemplo, ficheros de audio o vídeo.
res/values/	Ficheros XML que definen un determinado valor para definir un color, un estilo, una cadena de caracteres, etc. Se describen en la siguiente tabla.

Tabla 2: Tipos de recursos según carpeta en Android.

Veamos los tipos de recursos que encontramos dentro de la carpeta *values*:

Fichero por defecto `identificador`	**Descripción**
strings.xml `R.string`	Identifica cadenas de caracteres. `<string name="saludo">¡Hola Mundo!</string>`
colors.xml `R.color`	Un color definido en formato ARGB (alfa, rojo, verde y azul). Los valores se indican en hexadecimal en uno de los formatos: #RGB, #ARGB, #RRGGBB ó #AARRGGBB. `<color name="verde_opaco">#0f0</color>` `<color name=" red_translucido">#80ff0000</color>`
dimensions.xml `R.dimen`	Un número seguido de una unidad de medida. px – píxeles; mm – milímetros; in – pulgadas; pt – puntos (= 1/72 pulgadas); dp – píxeles independientes de la densidad (= 1/160 pulgadas); sp – igual que dp, pero cambia según las preferencias de tamaño de fuente. `<dimen name="alto">2.2mm</dimen>` `<dimen name="tamano_fuente">16sp</dimen>`
styles.xml `R.style`	Definen una serie de atributos que pueden ser aplicados a una vista o a una actividad. Si se aplican a una actividad, se conocen como temas. `<style name="TextoGrande" parent="@style/Text">` `<item name="android:textSize">20pt</item>` `<item name="android:textColor">#000080</item>` `</style>`

Fichero por defecto identificador	Descripción
`R.int`	Define un valor entero. `<integer name="max_asteroides">5</integer>`
`R.bool`	Define un valor booleano. `<bool name="misiles_ilimitados">true</bool>`
`R.id`	Define un recurso de *id* único. La forma habitual de asignar un *id* a los recursos es con el atributo `id="@+id/nombre"`. Aunque en algunos casos puede ser interesante disponer de un *id* previamente creado, para que los elementos así nombrados tengan una determinada función. Este tipo de id se utiliza en las vistas *TabHost* y *ListView*. `<item type="id" name="button_ok" />` `<item type="id" name="dialog_exit" />`
`R.array`	Una serie ordenada de elementos. Pueden ser de strings, de enteros o de recursos (TypedArray). `<string-array name="dias_semana">` `<item>lunes</item>` `<item>martes</item>` `</string-array>` `<integer-array name="primos">` `<item>2</item><item>3</item><item>5</item>` `</integer-array>` `<array name="asteroides">` `<item>@drawable/asteroide1</item>` `<item>@drawable/asteroide2</item>` `</array>`

***Tabla 3:** Tipos de recursos en carpeta values.*

Aunque el sistema crea ficheros que aparecen en la columna de la izquierda de la tabla anterior y se recomienda definir los recursos de cadena dentro de *strings.xml*, hay que resaltar que no es más que una sugerencia de organización. Sería posible mezclar cualquier tipo de recurso de esta tabla dentro de un mismo fichero y poner a este fichero cualquier nombre.



Vídeo[tutorial]: *Tipos de recursos en Android*

2.7.2. Acceso a los recursos

Una vez definido un recurso, este puede ser utilizado desde un fichero XML o desde Java. A continuación se muestra un ejemplo desde XML:

```
<ImageView
   android:layout_height="@dimen/alto"
   android:layout_width="match_parent"
   android:background="@drawable/asteroide"
   android:text="@string/saludo"
   android:text_color="@color/verde_opaco"/>
```


Para acceder a un recurso definido en los ejemplos anteriores, usaremos el siguiente código:

```
Resources res = getResources();
Drawable drawable = ContextCompat.getDrawable(R.drawable.asteroide);
String saludo = res.getString(R.string.saludo);
int color = ContextCompat.getColor(R.color.verde_opaco);
float tamanoFuente = res.getDimension(R.dimen.tamano_fuente);
int maxAsteroides = res.getInteger(R.integer.max_asteroides);
boolean ilimitados = res.getBoolean(R.bool.misiles_ilimitados);
String[] diasSemana = res.getStringArray(R.array.dias_semana);
int[] primos = res.getIntArray(R.array.primos);
TypedArray asteroides = res.obtainTypedArray(R.array.asteroides);
Drawable asteroide1 = asteroides.getDrawable(0);
```

```
val drawable = ContextCompat.getDrawable( R.drawable.asteroide)
val saludo = resources.getString(R.string.saludo)
val color = ContextCompat.getColor(R.color.verde_opaco)
val tamanoFuente = resources.getDimension(R.dimen.tamano_fuente)
val maxAsteroides = resources.getInteger(R.integer.max_asteroides)
val ilimitados = resources.getBoolean(R.bool.misiles_ilimitados)
val diasSemana = resources.getStringArray(R.array.dias_semana)
val primos = resources.getIntArray(R.array.primos)
val asteroides = resources.obtainTypedArray(R.array.asteroides)
val asteroide1 = asteroides.getDrawable(0)
```

2.7.3. Recursos del sistema

Además de los recursos que podamos añadir a nuestra aplicación, también podemos utilizar una serie de recursos que han sido incluidos en el sistema.



Vídeo[tutorial]: *Recursos del sistema en Android*

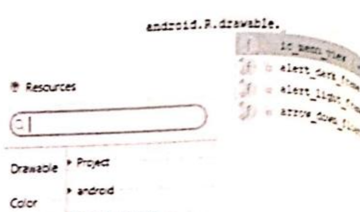
Usar recursos del sistema tiene muchas ventajas. No consumen memoria en nuestra aplicación, al estar ya incorporados al sistema. Además, los usuarios están familiarizados con ellos. Por ejemplo, si utilizamos el recurso `android.R.drawable.ic_menu_edit`, se mostrará al usuario el icono: ✎. Muy posiblemente, el usuario ya está familiarizado con este icono y lo asocia a la acción de editar. Otra ventaja es que los recursos del sistema se adaptan a las diferentes versiones de Android. Si se utiliza el tema `android.R.style.Theme_Panel`, este es bastante diferente en cada una de las versiones, pero seguro que estará en consonancia con el resto de estilos para esta versión. Lo mismo ocurre con el icono anterior. Este icono es diferente en algunas versiones, pero al usar un recurso del sistema nos aseguramos de que se mostrará el adecuado a la versión del usuario. Finalmente, estos recursos se adaptan siempre a las configuraciones locales.

Si yo utilizo el recurso `android.R.string.cancel`, este será "Cancelar", "Cancel", "消", etc., según el idioma escogido por el usuario.

Para acceder a los recursos del sistema desde código, usaremos la clase `android.R`. Se usa la misma estructura jerárquica de clases. Por ejemplo, `android.R.drawable.ic_menu_edit`. Para acceder desde XML, utilizamos la sintaxis habitual pero comenzando con `@android:`. Por ejemplo, `@android:drawable/ic_menu_edit`.

Para buscar recursos del sistema tienes varias alternativas:

- Usa la opción de autocompletar de Android Studio.
- Emplea el buscador de recursos que se incluye en el editor de *layouts*.
- Usa la aplicación `android.R` para explorar los recursos del sistema.



2.8. Estilos y temas

Si tienes experiencia con el diseño de páginas web, habrás advertido grandes similitudes entre HTML y el diseño de *layouts*. En los dos casos se utiliza un lenguaje de marcado y se trata de crear diseños independientes del tamaño de la pantalla donde se visualizarán. En el diseño web resultan clave las hojas de estilo en cascada (CSS), que permiten crear un patrón de diseño y aplicarlo a varias páginas. Cuando diseñes los *layouts* de tu aplicación, vas a poder utilizar unas herramientas similares conocidas como estilos y temas. Te permitirán crear patrones de estilo que podrán ser utilizados en cualquier parte de la aplicación. Estas herramientas te ahorrarán mucho trabajo y te permitirán conseguir un diseño homogéneo en toda tu aplicación.



Vídeo[tutorial]: *Estilos y temas en Android*

2.8.1. Los estilos

Un estilo es una colección de propiedades que definen el formato y la apariencia que tendrá una vista. Podemos especificar cosas como tamaño, márgenes, color, fuentes, etc. Un estilo se define en ficheros XML, diferente del fichero XML Layout que lo utiliza.

Veamos un ejemplo. El siguiente código:

```
<TextView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
```



```
    android:typeface="monospace"
    android:text="Un texto" />
```

Es equivalente a escribir:

```
<TextView
    style="@style/MiEstilo"
    android:text="Un texto" />
```

Habiendo creado en el fichero *res/values/styles.xml* con el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <style name="MiEstilo"
     parent="@android:style/TextAppearance.Medium">
       <item name="android:layout_width">match_parent</item>
       <item name="android:layout_height">wrap_content</item>
       <item name="android:textColor">#00FF00</item>
       <item name="android:typeface">monospace</item>
   </style>
</resources>
```

Observa como un estilo puede heredar todas las propiedades de un padre (*parámetro* `parent`) y a partir de estas propiedades realizar modificaciones.

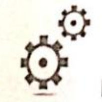
Heredar de un estilo propio

Si vas a heredar de un estilo definido por ti, no es necesario utilizar el atributo `parent`. Por el contrario, puedes utilizar el mismo nombre de un estilo ya creado y completar el nombre con un punto más un sufijo. Por ejemplo:

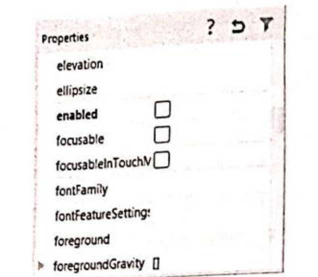
```
<style name="MiEstilo.grande">
     <item name="android:textSize">18pt</item>
</style>
```

Crearía un nuevo estilo que sería igual a `MiEstilo` más la nueva propiedad indicada. A su vez, puedes definir otro estilo a partir de este:

```
<style name="MiEstilo.grande.negrita">
     <item name="android:textStyle">bold</item>
</style>
```


Práctica: *Creando un estilo*

1. Abre el proyecto Asteroides o Mis Lugares.
2. Crea un nuevo estilo y llámalo `EstiloBoton`. Para ver las propiedades que puedes modificar te recomendamos que consultes la "Referencia de la clase View" en el anexo D. Para un botón puedes definir los atributos de `View`, `TextView` y `Button`. Otra alternativa consiste en seleccionar un botón en el editor visual de vistas y en la ventana *Properties* buscar las propiedades disponibles:



3. Aplícalo al primer botón del *layout*.
4. Crea un nuevo estilo y llámalo `EstiloBoton.Alternativo`. Este ha de modificar alguno de los atributos anteriores y añadir otros, como `padding`.
5. Aplícalo al segundo botón del *layout*.
6. Visualiza el resultado.

2.8.2. Los temas

Un tema es un estilo aplicado a toda una actividad o aplicación, en lugar de a una vista individual. Cada elemento del estilo solo se aplicará a aquellos elementos donde sea posible. Por ejemplo, `CodeFont` solo afectará al texto.

Para aplicar un tema a toda una aplicación, edita el fichero *AndroidManifest.xml* y añade el parámetro `android:theme` en la etiqueta `<application>`:

```
<application android:theme="@style/MiTema">
```

También puedes aplicar un tema a una actividad en concreto:

```
<activity android:theme="@style/MiTema">
```

Además de crear tus propios temas, vas a poder utilizar algunos disponibles en el sistema. Puedes encontrar una lista de todos los estilos y temas disponibles en Android en: http://developer.android.com/reference/android/R.style.html

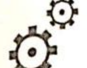


Ejercicio: *Aplicando un tema del sistema*

1. Abre el proyecto Asteroides o Mis Lugares.
2. Aplica a la actividad principal el tema `@style/Theme.AppCompat.Dialog` tal y como se acaba de mostrar.
3. Visualiza el resultado. Este tema es utilizado en cuadros de diálogo. No parece muy adecuado para nuestra actividad.



4. Deshaz el cambio realizado en este ejercicio.



Práctica: *Modificando el tema por defecto de la aplicación*

1. Abre el proyecto Asteroides o Mis Lugares.
2. Abre el fichero *res/values/styles.xml* (recurso por defecto).
3. Observa cómo se define el estilo `AppTheme` que será usado como estilo por defecto en la aplicación. Hereda de `Theme.AppCompat.Light.DarkActionBar` y solo define los colores principales usados en la aplicación. Añade las líneas subrayadas para personalizar un par de aspectos:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="android:typeface">serif</item>
    <item name="android:textColor">#0000FF</item>
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
</style>
```

4. Ejecuta la aplicación para visualizar el resultado.
5. Modifica otros atributos y comprueba el resultado.



Preguntas de repaso: *Estilos y temas*

2.9. Uso práctico de vistas y *layouts*

En este apartado vamos a aprender a usar varios tipos de vistas y *layouts* desde un punto de vista práctico. También empezaremos a escribir código que se ejecutará cuando ocurran ciertos eventos:



Ejercicio: *Un botón con gráficos personalizados*

1. Crea un nuevo proyecto con nombre *Mas Vistas* y tipo de actividad *Empty Activity*. Puedes dejar el resto de parámetros con los valores por defecto.
2. Crea el fichero *boton.xml* en la carpeta *res/drawable*. Para ello pulsa con el botón derecho sobre la carpeta *res/drawable y selecciona New > Drawable Resource File*. Introduce en el campo *File name:* «boton». Reemplaza el código por el siguiente:

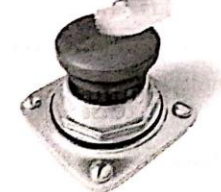
```
<?xml version="1.0" encoding="utf-8"?>
<selector
      xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:drawable="@drawable/boton_pulsado"
         android:state_pressed="true" />
   <item android:drawable="@drawable/boton_con_foco"
         android:state_focused="true" />
   <item android:drawable="@drawable/boton_normal" />
</selector>
```

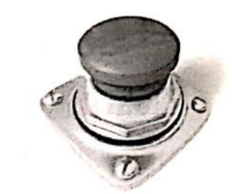
Este XML define un recurso único gráfico (*drawable*) que cambiará en función del estado del botón. El primer ítem define la imagen usada cuando se pulsa el botón, el segundo ítem define la imagen usada cuando el botón tiene el foco (cuando el botón está seleccionado con la rueda de desplazamiento o las teclas de dirección) y el tercero, la imagen en estado normal. Los gráficos, y en concreto los *drawables*, se estudiarán en el capítulo 4.

NOTA: El orden de los elementos <item> es importante. Cuando se va a dibujar se recorren los ítems en orden hasta que se cumpla una condición. Debido a que "boton_normal" es el último, solo se aplica cuando las condiciones state_pressed *y* state_focused *no se cumplen.*

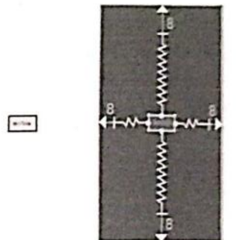
3. Descarga de http://www.androidcurso.com/index.php/119 las tres imágenes que aparecen a continuación. Para bajar cada imagen, pulsa sobre los nombres. Guárdalos con el nombre de fichero que se indica a continuación:


boton_normal.jpg


boton_con_foco. jpg


boton_pulsado. jpg

4. Selecciona los tres ficheros y cópialos en el portapapeles (*Ctrl-C*), selecciona la carpeta `res/drawable/` del proyecto y pega los ficheros (*Ctrl-V*). Te preguntará si quieres copiarlos a la carpeta de recursos por defecto a alguna de recursos alternativos. Selecciona la primera opción.
5. Abre el fichero `res/layout/activity_main.xml`.
6. En la ventana *Component Tree*, elimina el `TextView` que encontrarás dentro del `ConstraintLayout`.
7. Selecciona el `ConstraintLayout`. En la ventana *Attributes* busca el atributo `background`. Pulsa en el icono de la derecha y selecciona el recurso *Color/android/white*.
8. Arrastra una vista de tipo `Button` dentro del `ConstraintLayout`.
9. Sitúa el botón en el centro. Para ello, selecciona cada uno de sus puntos de anclaje arrastrando hasta el borde al que mira. El resultado ha de ser:



10. Selecciona el atributo `background` y pulsa el botón selector de recurso (con puntos suspensivos). Selecciona *Drawable/boton*.
11. Modifica el atributo `Text` para que no tenga ningún valor.
12. Introduce en el atributo `onClick` el valor `sePulsa`.

A continuación, se muestra el código resultante para *activity_main.xml*:

```
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity"
        android:background="@android:color/white">
    <Button android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/button"
            android:background="@drawable/boton"
            android:onClick="sePulsa"
            app:layout_constraintStart_toStartOf="parent"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"
            app:layout_constraintTop_toTopOf="parent"
            android:layout_marginBottom="8dp"
            app:layout_constraintBottom_toBottomOf="parent"
```

```
        android:layout_marginEnd="8dp"
        app:layout_constraintEnd_toEndOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

13. Abre el fichero *MainActivity* e introduce antes de la última llave, el código:

```
public void sePulsa(View view){
   Toast.makeText(this, "Pulsado", Toast.LENGTH_SHORT).show();
}
```

```
fun sePulsa(view: View) {
   Toast.makeText(this, "Pulsado", Toast.LENGTH_SHORT).show()
}
```

*NOTA: Pulsa **Alt-Intro** para que se añadan automáticamente los paquetes que faltan en la sección* `import`.

El método anterior se ejecutará cuando se pulse el botón. A este tipo de métodos se los conoce como escuchadores de eventos (*listeners*). Este método se limita a lanzar un *toast*, es decir, un aviso que permanece cierto tiempo sobre la pantalla y luego desaparece. Los tres parámetros son: el contexto utilizado, que coincide con la actividad, el texto a mostrar y el tiempo que permanecerá este texto. Los conceptos de *actividad* y *contexto* se desarrollarán en el siguiente capítulo.

14. Ejecuta el proyecto y verifica el resultado.

2.9.1. Acceder y modificar propiedades de las vistas por código



Ejercicio: *Acceder y modificar las propiedades de las vistas por código*

1. Abre el *layout activity_main.xml* creado en el ejercicio anterior.
2. En la paleta de vistas, dentro de *Text*, busca *Number (Decimal)* y arrástralo arriba del botón rojo.
3. Modifica algunos atributos de esta vista: `hint` = "Introduce un número", `ID` = "entrada".
4. Desde el marco *Palette/Common*, arrastra *Button* arriba del botón rojo.
5. Modifica algunos atributos de esta vista: Haz que su anchura ocupe toda la pantalla, que su texto sea "0" y que su *id* sea "boton0".
6. Busca ahora *TextView* y arrástralo debajo del botón rojo.
7. Modifica algunos atributos de esta vista: `TextColor` = #0000FF, `Text` = "", `Hint` = "Resultado", `id` = "salida".
8. Ajusta las restricciones de las vistas introducidas.

9. Abre el fichero *MainActivity*. **En Java**, vamos a añadir dos nuevas propiedades a la clase. Para ello copia el siguiente código al principio de la clase (antes del método `onCreate()`):

```
private EditText entrada;
private TextView salida;
```

*NOTA: Recuerda pulsar **Alt-Intro** para que se añadan los paquetes de las dos nuevas clases utilizadas.*

10. **En Java**, copia al final del método `onCreate()` las siguientes líneas:

```
entrada = findViewById(R.id.entrada);
salida = findViewById(R.id.salida);
```

Como se explicó al principio del capítulo, las diferentes vistas definidas en *activity_main.xml* son creadas como objetos Java cuando se ejecuta `setContentView(R.layout.activity_main)`. Si queremos manipular algunos de estos objetos hemos de declarar las variables (paso 9) y asignarles la referencia al objeto correspondiente (paso 10). Para ello, hay que introducir el atributo `id` en XML y utilizar el método `findViewById(R.id.valor_en_atributo_id)`. Este método devuelve un objeto de la clase `View`.

En Kotlin este proceso se realiza automáticamente. Lo único que tienes que hacer es asegurarte que se ha importado el paquete `kotlinx.android.synthetic.main.activity_main.*`.

11. Introduce en el atributo `onClick` del botón con *id* `boton0` el valor "sePulsa0". De esta manera, cuando se pulse sobre el botón se ejecutará el método `sePulsa0()`. Según la jerga de Java, diremos que este método es un escuchador del evento *click* que puede generar el objeto `boton0`.
12. Añade el siguiente método al final de la clase `MainActivity`.

```
public void sePulsa0(View view){
   entrada.setText(entrada.getText()+"0");
}
```

```
fun sePulsa0(view: View) {
   entrada?.setText(entrada?.text.toString() + "0")
}
```

Lo que hace es asignar como textos de `entrada` el resultado de concatenar al texto de `entrada` el carácter "0". *NOTA: Si eres nuevo en Kotlin, te habrá extrañado el uso de ?. Lee la sección* Tratamiento de null en Kotlin [19].

13. Añade al botón con texto "0" el atributo *tag* = "0".
14. Modifica el método `sePulsa0()` de la siguiente forma:

```
entrada.setText(entrada.getText()+(String)view.getTag());
```

```
entrada?.setText(entrada?.getText().toString() + view?.tag as String)
```

[19] http://www.androidcurso.com/index.php/922

El resultado obtenido es equivalente al anterior. En algunos casos será interesante utilizar un mismo método como escuchador de eventos de varias vistas. Podrás averiguar la vista que causó el evento, dado que esta se pasa como parámetro del método. En el ejemplo sabemos que en el atributo `tag` guardamos el carácter a insertar. El atributo `tag` puede ser usado libremente por el programador para almacenar un objeto de la clase `Object` (en la práctica podemos usar cualquier tipo de clase, dado que `Object` es la clase raíz de la que heredan todas las clases en Java). En nuestro caso hemos almacenado un objeto `String`, por lo que necesitamos una conversión de tipo. *NOTA: Utiliza esta forma de trabajar en la práctica para no tener que crear un método* onClick *para cada botón.*

15. Modifica el código de `sePulsa()` con el siguiente código:

```
salida.setText(String.valueOf(Float.parseFloat(
        entrada.getText().toString()) * 2.0));
```

```
salida?.setText((java.lang.Float.parseFloat(
        entrada?.getText().toString()) * 2.0).toString())
```

En este código el valor de `entrada` es convertido en `Float`, multiplicado por dos y convertido en `String` para ser asignado a `salida`.

16. Ejecuta el proyecto y verifica el resultado. *NOTA: En este ejercicio no se ha realizado la verificación de que los datos introducidos por el usuario. Has de tener introducir datos válidos y en el orden adecuado.*



Preguntas de repaso: *Uso práctico de Vistas*

2.10. Uso de *tabs* (pestañas)

Los *tabs* nos van a permitir crear una interfaz de usuario basada en pestañas, donde, de una forma muy intuitiva, podemos ofrecer al usuario diferentes contenidos, que son seleccionados al pulsar una de las pestañas que se muestran en la parte superior.



En este apartado usaremos la clase `TabLayout` para crear pestañas, aunque existen otras alternativas anteriores, como `TabHost` y `FragmentTabHost`, que han sido declaradas como obsoletas.

La vista `TabLayout` es la nueva alternativa propuesta en Material Design. Está disponible en la librería de compatibilidad Design Support Library, por lo que podremos utilizarla desde versiones antiguas de Android. A continuación, se muestra el esquema a utilizar en XML:

```
<com.google.android.material.tabs.TabLayout
    android:id="@+id/tabs"
```



```
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```



Ejercicio: *Añadir pestañas con TabLayout*

1. Crea un nuevo proyecto de tipo *Empty Activity* y con nombre *Tabs*. La versión mínima de API ha de ser como mínimo 17.
2. Reemplaza el contenido de *activity_main.xml* por el siguiente:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tabs"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@+id/texto"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center"
        android:text="¡Hola Mundo!" />
</LinearLayout>
```

3. La clase `TabLayout` ha sido añadida en la librería de compatibilidad de Material Design. Para poder utilizarla, añade la siguiente dependencia en *build.gradle (Module: app)*:

```
implementation 'com.google.android.material:material:1.2.1'
```

4. Al final del método `onCreate()` de `MainActivity` añade el siguiente código:

```
//Pestañas
TabLayout tabs = findViewById(R.id.tabs);
tabs.addTab(tabs.newTab().setText("Pestaña 1"));
tabs.addTab(tabs.newTab().setText("Pestaña 2"));
tabs.addTab(tabs.newTab().setText("Pestaña 3"));
```

Comenzamos buscando la vista `TabLayout` del Layout. Las siguientes tres sentencias permiten insertar tres pestañas. Aparecerán en el orden en que se añaden usando el método `addTab()`. El método `newTab()` crea la nueva pestaña, que es configurada a continuación, asignándole un texto.

Ejecuta el proyecto y verifica que aparecen las tres pestañas.

5. Para darle funcionalidad a las pestañas añade a continuación este código:

```
final TextView texto = findViewById(R.id.texto);
tabs.addOnTabSelectedListener(new TabLayout.OnTabSelectedListener() {
    @Override public void onTabSelected(TabLayout.Tab tab) {
```

```
        switch (tab.getPosition()) {
            case 0:
                texto.setText("Pestaña 1");
                break;
            case 1:
                texto.setText("Pestaña 2");
                break;
            case 2:
                texto.setText("Pestaña 3");
                break;
        }
    }
    @Override public void onTabUnselected(TabLayout.Tab tab) {}
    @Override public void onTabReselected(TabLayout.Tab tab) {}
});
```

Comenzamos buscando el `TexView` que queremos modificar. Luego, asignado un escuchador de eventos al `TabLayout` formado por tres métodos. Los nombres de los métodos nos indican claramente cuándo van a ser llamados. Los tres tienen como parámetro la pestaña que produce el evento (`Tab`). De los tres métodos solo introducimos código en el primero: Según la posición de la etiqueta seleccionada, modificamos el texto del `TexView`.

Ejecuta el proyecto y verifica el resultado.



Ejercicio: *Controlar pestañas con ViewPager2*

Aunque no es el caso del ejemplo anterior, en la mayoría de las ocasiones las pestañas se usan para asignar a cada una de ellas un *fragment*, de forma que, al ser pulsadas se visualiza un *fragment* diferente. Para realizar este proceso de forma más sencilla vamos a utilizar un `ViewPager2`, que como su nombre sugiera se trata de una versión mejorada de `ViewPager`. La finalidad de un `ViewPager` es visualizar una serie de páginas, donde el usuario puede navegar arrastrando a derecha o izquierda. Cada página ha de estar definida en un fragment. Veamos cómo hacerlo.

1. En *activity_main.xml* reemplaza el `TextView` por el siguiente código:

```
<androidx.viewpager2.widget.ViewPager2
    android:id="@+id/viewpager"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

En este `ViewPager2` se visualizará un fragment diferente para cada pestaña.

2. La clase `ViewPager2` ha sido añadida en la librería de copativilidad. Para poder utilizarla, añade la siguiente dependencia en *build.gradle (Module: app)*:

```
implementation 'androidx.viewpager2:viewpager2:1.0.0'
```

3. Dentro de `MainActivity` añade el siguiente código al final (justo antes de }):


```
public class MiPagerAdapter extends FragmentStateAdapter {

    public MiPagerAdapter(FragmentActivity activity){
        super(activity);
    }

    @Override
    public int getItemCount() {
        return 3;
    }

    @Override @NonNull
    public Fragment createFragment(int position) {
        switch (position) {
            case 0: return new Tab1();
            case 1: return new Tab2();
            case 2: return new Tab3();
        }
        return null;
    }
}
```

Esta clase es un adaptador (`Adapter`), es un mecanismo muy utilizado en Android para hace de puente entre nuestros datos y el interfaz de usuario. Más adelante lo utilizaremos para rellenar los datos de un `ReciclerView` o un `Spinner`. Para este ejemplo, lo que hace es indicar cuantos tabs queremos (`getCount()`) y que `Fragment` visualizamos en cada tab (`createFragment(int)`).

4. En `onCreate()` reemplaza todo el código creado para los tabs por:

```
//Pestañas
ViewPager2 viewPager = findViewById(R.id.viewpager);
viewPager.setAdapter(new MiPagerAdapter(this));
TabLayout tabs = findViewById(R.id.tabs);
new TabLayoutMediator(tabs, viewPager,
    new TabLayoutMediator.TabConfigurationStrategy() {
        @Override
        public void onConfigureTab(@NonNull TabLayout.Tab tab, int position){
                tab.setText(nombres[position]);
        }
    }
).attach();
```

Comenzamos obteniendo el `ViewPager2` y la asignamos el adaptador. Buscamos también la vista `tabs`. Luego creamos un objeto `TabLayoutMediator` para asignar los `tabs` al `viewPager` y para indicar los nombres de cada `tabs`.

5. Añade el siguiente atributo a la clase:

```
// Nombres de las pestañas
private String[] nombres = new String[]{"Pestaña 1","Pestaña 2","Pestaña 3"};
```

6. Crea un nuevo *layout* y llámalo *tab1.xml*:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
          android:layout_width="match_parent"
          android:layout_height="match_parent"
          android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              android:gravity="center_vertical|center_horizontal"
              android:text="Pestaña 1"
              android:textAppearance=
                        "?android:attr/textAppearanceMedium" />
</LinearLayout>
```

7. Crea una nueva clase con nombre *Tab1.java*:

```
public class Tab1 extends Fragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
              Bundle savedInstanceState) {
       return inflater.inflate(R.layout.tab1, container, false);
    }
}
```

```
class Tab1 : Fragment() {
   override fun onCreateView(inflater: LayoutInflater,
             container: ViewGroup?, savedInstanceState: Bundle?): View? {
      return inflater?.inflate(R.layout.tab1, container, false)
   }
}
```

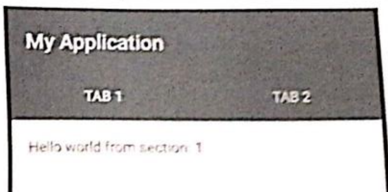
Un *fragment* se crea de forma muy parecida a una actividad. También dispone del método `onCreate()`. En este ejemplo se llama al mismo método del antecesor, sin introducir nuevo código. Un *fragment* también tiene asociada una vista, aunque a diferencia de una actividad, no se asocia en el método `onCreate()`, si no que dispone de un método especial para esta tarea: `onCreateView()`. Dentro de este método vamos a utilizar un `LayoutInflater` para, a partir de un layout en XML, crear un objeto de la clase `View`. Para ello usaremos su método `inflate()`, que dispone de tres parámetros. El recurso de layout, el contenedor donde se tiene previsto insertar el layout y cuándo queremos insertarlo: ya mismo (`true`), o si de momento no queremos insertarlo (`false`).

8. Repite los dos pasos anteriores para crear *tab2.xml* y *Tab2.java*.
9. Repite de nuevo para crear el *layout tab3.xml* y la clase *Tab3.java*.
10. Modifica estos ficheros para que cada *layout* sea diferente y para que cada *fragment* visualice el *layout* correspondiente.
11. Ejecuta el proyecto.

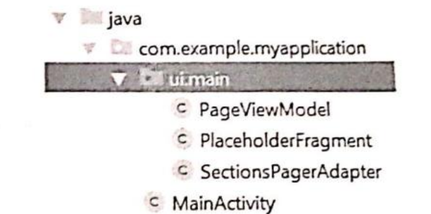


Ejercicio: *Añadir pestañas con TabLayout de forma automática*

1. Crear un nuevo proyecto y seleccionar como actividad inicial Tabbed Activity. Si ya dispones de un proyecto y quieres añadir una actividad con pestañas utiliza la opción *File / New / Activity / Tabbed Activity*.
2. Ejecuta el proyecto y verifica que el resultado es parecido al ejercicio anterior:



3. En el explorador del proyecto observa como, además de MainActivity, se han creado tres clase tres clases en el paquete *com.example.myapplication.ui.main*:



Se ha creado este subpaquete para dar a entender que estas tres clases son utilizadas por la actividad main.

`SectionsPagerAdapter` tendría una función similar a `MiPagerAdapter`. Es decir, indicar cuantas pestañas queremos, sus nombres y los `Fragments` para cada una.

`PlaceholderFragment` es el fragment que pondremos dentro de las pestañas. En lugar de crear tres fragments diferentes (`Tab1`, `Tab2` y `Tab3`) tendremos una única clase a la que pasaremos un valor entero según la pestaña a crear. Cuando vaya a crear la vista utilizará este valor entero para personalizarla. Realmente la personalización se hace a través de la siguiente clase.

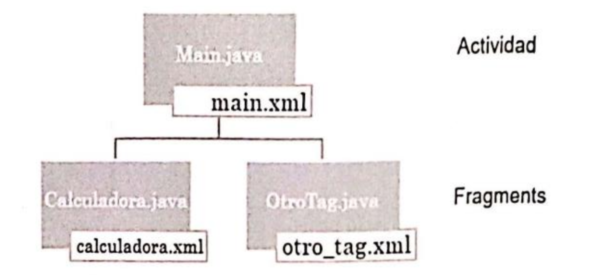
`PageViewModel` se basa en `ViewModel` y permite almacenar datos relacionados con la interfaz de usuario. Su principal función es que los datos sobrevivan a los cambios de configuración, como las rotaciones de pantalla.

4. Trata de modificar el proyecto para que se generen tres pestañas en lugar d dos.

2.10.1. Evento onClick en un Fragment

Trabajar con una actividad que contiene varios fragments, suele ocasionar ciertos errores en la programación que se repiten con mucha frecuencia. Para ayudarte a realizar este trabajo correctamente, hemos introducido este apartado.

Supongamos que tenemos la actividad Main.java que visualiza el layout main.xml. Dentro de esta actividad se pueden visualizar dos fragments, aunque no de forma simultánea. Por ejemplo, se podría visualizar uno u otro usando un `TabLayout`. El siguiente esquema muestra los nombres de los fragments y los layouts que estos visualizan.



Vamos a suponer que dentro del layout calculadora se ha añadido en uno de sus botones un atributo `onClick`:

```
<Button … onClick="sePulsa">
```

Importante: el método `sePulsa(View v)` hay que declararlo en la actividad. Es en la actividad donde lo va a buscar, si lo pones en otro sitio no lo encontrará.

En este punto se nos plantea una cuestión: ¿Dónde poner el código que gestiona el comportamiento de un fragment? ¿En la clase de la actividad (Main.java)? ¿O en la del fragment (Calculadora.java)? La respuesta puede variar según el contexto, pudiéndose dar dos casos principales:

Los fragments son parte del mismo proceso

En el caso de que tanto la actividad como los fragments son parte del mismo proceso, podemos centralizar todo el código en la actividad. Dividir el código en varias clases, si se trata de un código con una sola función, puede ser confuso. El código de cada fragment solo tendrá que mostrar el *layout* correspondiente.

En este supuesto puede darse el caso de que queramos acceder a una vista definida dentro del layout de un fragment. Resulta frecuente querer disponer de esta vista en un objeto, e inicializarlo en el método `onCreate()`. Pero cuidado, si lo intentamos hacer en el método `onCreate()` de la actividad nos dará un error. En este método tras llamar al `super`, podemos suponer que la actividad ya está creada y también su layout. Sin embargo, los diferentes fragments no están todavía creados, por lo que al tratar de acceder a una de sus vistas, va a dar un error. Para resolver el problema podemos usar el siguiente código:


```
public class Main extends Activity {
  EditText cantidad;

  @Override public void onCreate(...) {
    super.onCreate(...);
    cantidad = findViewById(R.id.editText1); //NO FUNCIONA
  }

  public void sePulsa(View view) {
    if (cantidad == null) {
      cantidad = findViewById(R.id.editText1);
    }
    cantidad.setText(cantidad.getText() + (String) view.getTag());
  }
}
```

Lo que hacemos es posponer la creación del objeto `cantidad` hasta que sepamos seguro que el fragment donde aparece ha sido creado. En el ejemplo, si el método `sePulsa()` está asociado a un botón que aparece en el mismo fragment, cuando se ejecute el método podemos estar seguros que el fragment está creado. Como solo hace falta crearlo una vez, comparamos si en la variable hay `null` para crearlo. Si no, es que ya está creado.

Los fragments son independientes

Se puede dar el caso de que cada fragment tenga una función diferente. Por ejemplo, en uno hay una calculadora y en otro un diccionario. En este caso es importante que el código de cada fragment se escriba en su clase. Si actuamos de esta forma, podremos reutilizar el fragment en otra aplicación. Por ejemplo, si queremos añadir una calculadora en otra aplicación, no tendremos más que incluir su clase Calculadora.java con su layout.

Aquí aparece un problema, si queremos controlar la pulsación de un botón de la calculadora y lo hacemos con el atributo `onClick`, el método llamado ha de estar en la actividad. Esto rompe el principio de que todo el código que controle el fragment ha de estar en su clase.

Para resolver el problema vamos a utilizar un método alternativo, que consiste en programar un escuchador de evento por código. Veamos un ejemplo:

```
public class Calculadora extends Fragment {
  EditText cantidad;
  ...
  @Override public View onCreateView(...) {
    View v = inflater.inflate(R.layout.calculadora,container,false);
    cantidad = v.findViewById(R.id.editText1);
    Button boton = v.findViewById(R.id.Button00);
    boton.setOnClickListener(new OnClickListener() {
      public void onClick(View view) {
        sePulsa(view);
      }});   …
  }
```

```
  public void sePulsa(View view) {
    cantidad.setText(cantidad.getText() + (String) view.getTag());
  }
}
```

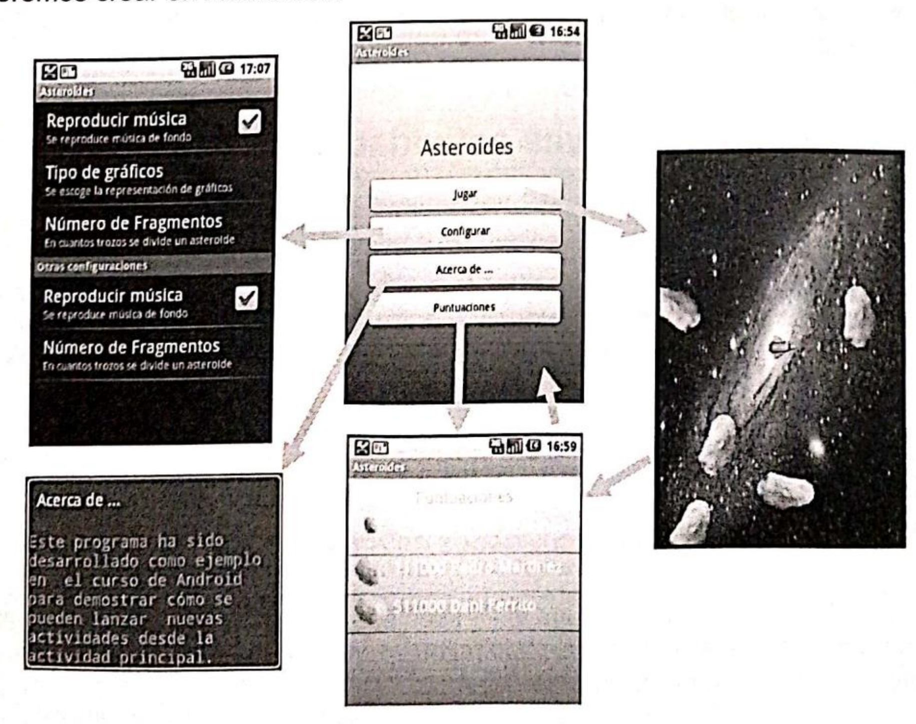
Si lo comparamos con el caso anterior, la asignación del layout al fragment se realiza de forma diferente. Ahora usamos en método `onCreateView()`, en lugar de `onCreate()`. El layout es creado en la vista, `v`, usando un el inflador, `inflater`. Recuerda que un inflador nos convierte un fichero XML a su correspondiente objeto Java. Llamando a `v.findViewById()` podemos extraer vistas concretas del layout.

Tras extraer `boton`, le asociamos un escuchador de evento `onClick` llamando a `setOnClickListener()`. Para crear un escuchador de eventos en Java, hay que crear un objeto de la clase adecuada `OnClickListener` y escribir un método para cada uno de los eventos que el botón puede generar. En nuestro caso solo el método `onClick(View)`. El parámetro que recibe el método corresponde al objeto que lanzo el evento. Tal y como se ha escrito el código solo puede ser `botón`.

CAPÍTULO 3.
Actividades e intenciones

En este capítulo seguiremos trabajando con el diseño de la interfaz de usuario. En lugar de tratar aspectos de diseño visual, como hemos hecho en el capítulo anterior, vamos a centrarnos en temas más relacionados con el código. En concreto, nos centraremos en las *actividades* y las *intenciones.* Estudiaremos también dos herramientas de gran utilidad para cualquier aplicación: la barra de acciones y la definición de las preferencias de configuración. Además, se tratará un tipo de vista muy práctica, aunque algo compleja de manejar: `RecyclerView`.

Nos vamos a centrar en los dos ejemplos de aplicaciones que estamos desarrollando, Asteroides y Mis Lugares, para añadirle diferentes actividades. A continuación, se muestra el esquema de navegación entre las actividades que queremos crear en Asteroides.



A medida que crece el tamaño de una aplicación, resulta más complicado mantener el código ordenado y sin errores. Es aquí donde el uso de arquitecturas correctas de *software* puede ayudarnos. En este capítulo vamos a realizar una breve introducción de estos conceptos. En concreto, veremos cómo almacenar información que pueda ser accesible desde toda la aplicación y la arquitectura Clean. El objetivo principal no es adquirir unos conocimientos profundos en esta materia, sino que un programador novel asuma la importancia de utilizar estas técnicas utilizándolas en la práctica.



Objetivos:

- Describir el conjunto de actividades que forman la interfaz de usuario en una aplicación Android.
- Mostrar cómo podemos, desde una actividad, invocar a otras y cómo podemos comunicarnos con ellas.
- Incorporar a nuestras aplicaciones ciertos elementos prácticos, tales como los menús o las preferencias.
- Introducir conceptos de arquitectura de *software*, como el patrón Singleton y la arquitectura Clean.
- Describir cómo podemos utilizar y crear iconos en nuestras aplicaciones.
- Estudiar una vista para crear listas en Android: `RecyclerView`.
- Describir el uso de intenciones para invocar actividades estándar en Android.

3.1. Creación de nuevas actividades

El concepto de actividad en Android representa una unidad de interacción con el usuario. Corresponde a lo que coloquialmente llamamos una pantalla de la aplicación. Una aplicación suele estar formada por una serie de actividades, de forma que el usuario puede ir navegando entre actividades. En concreto, Android suele disponer de un botón (físico o en pantalla) que nos permite volver a la actividad anterior.



Vídeo[tutorial]: *Actividades en Android*

Toda actividad ha de tener una vista asociada, que será utilizada como interfaz de usuario. Esta vista suele ser de tipo *layout*, aunque también puede ser una vista simple, como se verá en el siguiente ejemplo.

Una aplicación estará formada por un conjunto de actividades independientes; es decir, se trata de clases independientes que no comparten variables, aunque todas trabajan para un objetivo común. Otro aspecto importante es que toda actividad ha de ser una subclase de `Activity`.

Las aplicaciones creadas en los ejemplos hasta ahora disponían de una única actividad, que se creaba automáticamente y a la que se asignaba la vista definida en *res/layout/activity_main.xml*. Esta actividad era arrancada al comenzar la aplicación. A medida que nuestra aplicación crezca, será imprescindible crear nuevas actividades. En este apartado describiremos cómo hacerlo. Este proceso se puede resumir en cuatro pasos:

- Crear un nuevo *layout* para la actividad.
- Crear una nueva clase descendiente de `Activity`. En esta clase tendrás que indicar que el *layout* a visualizar es el desarrollado en el punto anterior.
- Para que nuestra aplicación sea visible, será necesario activarla desde otra actividad.
- De forma obligatoria tendremos que registrar toda nueva actividad en *AndroidManifest.xml*.

Veamos un primer ejemplo de cómo crear una nueva actividad en la aplicación que estamos desarrollando.



Ejercicio: *Implementación de una caja Acerca de*

Vamos a crear una caja *Acerca de...* y a visualizarla cuando se pulse el botón adecuado. Puedes realizarlo tanto en Asteroides como en Mis Lugares.

1. En primer lugar, crea el fichero *res/layout/acercade.xml*. Para ello pulsa con el botón derecho sobre el explorador del proyecto en la carpeta *res/layout* y selecciona *New > Layout resource file*. Indica en *File name: acercade*.
2. Selecciona la lengüeta de edición en Text y copia el siguiente contenido:

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:padding="@dimen/text_margin"
    android:text="Este programa ha sido desarrollado como ejemplo en el
curso de Android para demostrar cómo se pueden lanzar nuevas actividades
desde la actividad principal.">
</TextView>
```

3. Creamos ahora una nueva actividad, que será la responsable de visualizar esta vista. Para ello crea el fichero *AcercaDeActivity.java* pulsando con el botón

derecho sobre el nombre del paquete de la aplicación y seleccionando *New > Java Class*. En el campo *Name* introduce *AcercaDeActivity* y pulsa *Finish*. Reemplaza el código por:

```
public class AcercaDeActivity extends AppCompatActivity {
   @Override public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.acercade);
   }
}
```

```
class AcercaDeActivity : AppCompatActivity() {
   public override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.acercade)
   }
}
```

Nota sobre Java/Kotlin: *Pulsa Alt-Intro en las dos clases modificadas para que automáticamente se añadan los paquetes que faltan.*

4. Pasemos ahora a crear un método en la actividad principal que se ejecutará cuando se pulse el botón *Acerca de*.

```
public void lanzarAcercaDe(View view){
   Intent i = new Intent(this, AcercaDeActivity.class);
   startActivity(i);
}
```

```
fun lanzarAcercaDe(view: View? = null) {
   val i = Intent(this, AcercaDeActivity::class.java)
   startActivity(i)
}
```

5. Para asociar este método al botón, edita el *layout activity_main.xml* (o *content_main.xml* en Mis Lugares). Selecciona la lengüeta *Design* y pulsa sobre el botón *Acerca de…* y en la vista *Properties* busca el atributo `onClick` e introduce el valor `lanzarAcercaDe`.

6. Selecciona la lengüeta *Code* y observa cómo, en la etiqueta `<Button>` correspondiente, se ha añadido el atributo:

   ```
   android:onClick="lanzarAcercaDe"
   ```

   NOTA: En caso de que exista algún recurso alternativo para el layout, repite el mismo proceso.

7. Ejecuta ahora la aplicación y pulsa el botón *Acerca de*. Observarás que el resultado no es satisfactorio. ¿Qué ha ocurrido?

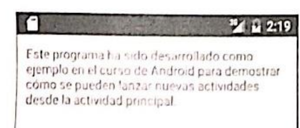
Se ha detenido la aplicación Mis Lugares.

ACEPTAR

El problema es que toda actividad que ha de ser lanzada por una aplicación ha de ser registrada en el fichero *AndroidManifest.xml*. Para registrar la actividad, abre *AndroidManifest.xml*. Añade el siguiente texto dentro de la etiqueta `<application …> </ application>`:

```
<activity android:name=".AcercaDeActivity"
          android:label="Acerca de ..."/>
```

8. Ejecuta de nuevo el programa. El resultado ha de ser similar al mostrado a continuación:

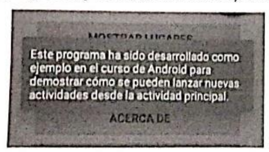


La vista mostrada en el ejemplo anterior no parece muy atractiva. Tratemos de mejorarla aplicando un tema. Como vimos en el capítulo anterior, un tema es una colección de estilos que define el aspecto de una actividad o aplicación. Puedes utilizar alguno de los temas disponibles en Android o crear el tuyo propio.

9. En este caso utilizaremos uno de los de Android. Para ello abre *AndroidManifest.xml* e introduce la línea subrayada:

```
<activity android:name=".AcercaDeActivity"
          android:label="Acerca de ..."
          android:theme="@style/Theme.AppCompat.Light.Dialog"/>
```

10. Ejecuta de nuevo el programa y observa como la apariencia mejora:



Ejercicio: *Un escuchador de evento por código*

Como acabamos de ver, en un *layout* podemos definir el atributo XML `onClick`, que nos permite indicar un método que se ejecutará al hacer clic en una vista. A este método se le conoce como escuchador de evento. Resulta muy sencillo y

además está disponible en cualquier descendiente de la clase `View`. Sin embargo, esta técnica presenta dos inconvenientes: no se puede usar con `Fragments` y solo está disponible para el evento `onClick()`. La clase `View` tiene otros eventos (`onLongClick()`, `onFocusChange()`, `onKey()`, etc.) para los que no se ha definido un atributo XML. Entonces, ¿qué hacemos si queremos definir un evento distinto de `onClick()`? La respuesta la encontrarás en este ejercicio:

1. Abre la clase *MainActivity (en Mis Lugares MainActivity)* y añade las líneas que aparecen subrayadas:

```
public class MainActivity extends Activity {
    private Button bAcercaDe;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        bAcercaDe = findViewById(R.id.button03);
        bAcercaDe.setOnClickListener(new OnClickListener() {
              public void onClick(View view) {
                 lanzarAcercaDe(null);
              }
        });
    }
    …
```

```
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      button03.setOnClickListener{
         lanzarAcercaDe()
      }
      …
```

Nota sobre Java/Kotlin: *Pulsa **Alt-Intro** en las dos clases modificadas para que automáticamente se añadan los paquetes que faltan. Para la clase OnClickListener selecciona android.view.View.OnClickListener.*

2. Elimina el atributo añadido al botón:

   `android:onClick="lanzarAcercaDe"`

3. Ejecuta la aplicación. El resultado ha de ser idéntico al anterior.

   NOTA: *En el capítulo 5 se estudiarán con más detalle los escuchadores de evento.*



Práctica: *El botón* Salir

En el *layout activity_main.xml* (o *content_main.xml* en Mis Lugares) hemos introducido un botón con el texto "Salir". Queremos que cuando se pulse este botón, se cierre la actividad. Para cerrar una actividad puedes llamar al método `finish();`. Llamar a este método es equivalente a pulsar la tecla "retorno".

1. Realiza este trabajo utilizando un escuchador de evento por código.
2. Hazlo ahora con el atributo XML `android:onClick`.
3. Verifica que el resultado es el mismo en ambos casos.

   NOTA: *No es conveniente que en tus actividades incluyas un botón para cerrarlas. Un dispositivo Android siempre dispone de la tecla "retorno", que tiene la misma función.*



Solución:

1. Para resolverlo mediante un escuchador por código, añade en el método `onCreate()` de la clase `MainActivity` el siguiente código:

```
Button bSalir = findViewById(R.id.button04);
bSalir.setOnClickListener(new OnClickListener() {
    public void onClick(View view) {
        finish();
    }
});
```

```
Button04.setOnClickListener{
   finish()
}
```

2. Para resolverlo con el atributo `onClick`, añade en `MainActivity` el método:

```
public void salir(View view){
    finish();
}
```

```
fun salir(view: View?) {
    finish();
}
```

Y añade el siguiente atributo al botón *Salir* en el *layout activity_main.xml*:

```
android:onClick="salir"
```



Preguntas de repaso: *Actividades*

3.2. Comunicación entre actividades

Cuando una actividad ha de lanzar a otra actividad, en muchos casos necesita enviarle cierta información.



Vídeo[tutorial]: *Intercambio de datos entre actividades*

Android nos permite este intercambio de datos utilizando el mecanismo que se describe a continuación:

Cuando lances una actividad B, desde la actividad A, usa el siguiente código en A:

```
Intent intent = new Intent(this, MI_CLASE.class);
intent.putExtra("usuario", "Pepito Perez");
intent.putExtra("edad", 27);
startActivity(intent);
```

```
val intent = Intent(this, MI_CLASE::class.java)
intent.putExtra("usuario", "Pepito Perez")
intent.putExtra("edad", 27)
startActivity(intent)
```

En la actividad lanzada (B) podemos recoger los datos de la siguiente forma:

```
Bundle extras = getIntent().getExtras();
String s = extras.getString("usuario");
int i = extras.getInt("edad");
```

```
val extras = intent.extras
val s = extras?.getString("usuario")?:"sin usuario"
val i = extras?.getInt("edad")?:-1
```

Cuando la actividad lanzada (B) termina, si lo desea, podrá enviar datos de vuelta. Para ello añade en la actividad B el siguiente código:

```
Intent intent = new Intent();
intent.putExtra("resultado","valor");
setResult(RESULT_OK, intent);
finish();
```

```
val intent = Intent()
intent.putExtra("resultado", "valor")
setResult(Activity.RESULT_OK, intent)
finish()
```

Es posible que el trabajo realizado en la actividad B sea cancelado. Para este caso añade:

```
Intent intent = new Intent();
setResult(RESULT_CANCEL, intent);
finish();
```



```
val intent = Intent()
setResult(Activity.RESULT_CANCEL, intent)
finish()
```

En la actividad que hizo la llamada (A) has de poder recoger estos datos. Para ello tendremos que lanzar la actividad con `startActivityForResult(Intent, int)` en lugar de `startActivityForResult(Intent)`, donde el segundo parámetro es un entero con un código que identifica a la actividad que lanzamos. Además, tendremos que sobrescribir el método `onActivityResult(int, int, Intent)`, donde en el primer parámetro se devuelve el mismo código que indicamos cuando hicimos la llamada. El segundo, si el resultado ha sido ok o cancelado. Y el tercero, un `Intent` donde se incluyen las variables devueltas:

```
Intent intent = new Intent(this, MI_CLASE.class);
startActivityForResult(intent, 1234);

@Override protected void onActivityResult(int requestCode, int resultCode,
                                          Intent data){
   super.onActivityResult(requestCode, resultCode, data);
   if (requestCode==1234 && resultCode==RESULT_OK) {
      String res = data.getExtras().getString("resultado");
   }
}
```

```
val intent = Intent(this, MI_CLASE::class.java)
startActivityForResult(intent, 1234)

override fun onActivityResult(requestCode: Int, resultCode: Int,
                              data: Intent?) {
   if (requestCode == 1234 && resultCode == Activity.RESULT_OK) {
      val res = data?.extras?.getString("resultado")?:"sin resultado"
   }
}
```

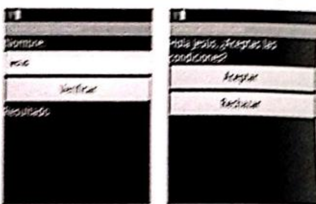
Desde la actividad A se podrían llamar a varias actividades. Sin embargo, solo podemos tener un método `onActivityResult()`. Por esta razón, resulta necesario identificar cada actividad lanzada con un código. Así podremos diferenciar entre los distintos datos devueltos.



Práctica: *Comunicación entre actividades*

1. Crea un nuevo proyecto con nombre *Comunicación Actividades* y tipo *Empty Activity*.
2. El *layout* de la actividad inicial ha de ser similar al que se muestra abajo a la izquierda.
3. Introduce el código para que cuando se pulse el botón *Verificar* se arranque una segunda actividad. A esta actividad se le pasará como parámetro el nombre introducido en el `EditText`.

4. El layout correspondiente a la segunda actividad se muestra a la derecha.
5. Al arrancar la actividad, el texto del primer `TextView` ha de modificarse para que ponga "Hola +nombre recibido+, ¿Aceptas las condiciones?"
6. En esta actividad se podrán pulsar dos botones, de forma que se devuelva a la actividad principal el `String` «Aceptado», al pulsar en Aceptar. Al pulsar el botón Cancelar también se regresará a la actividad anterior.
7. En la actividad principal se modificará el texto del último `TextView` para que ponga «Resultado: Aceptado» o «Resultado: Rechazado», según lo recibido.





Preguntas de repaso: *Comunicación entre Actividades*

3.3. La barra de acciones (*Toolbar*)



Video[tutorial]: *La barra de acciones (Toolbar)*

Desde la versión 3.0, se introdujo en Android un nuevo elemento en la interfaz de usuario: la barra de acciones. Esta se sitúa en la parte superior de la pantalla, fue creada para que el usuario tuviera una experiencia unificada a través de las distintas aplicaciones. La barra de acciones aglutina varios elementos; los más habituales son el nombre de la aplicación, el botón para abrir el *Navigation Drawer* y los botones de acciones frecuentes. Las acciones menos utilizadas se sitúan en un menú desplegable, que se abrirá desde el botón *Overflow* (se representa con tres puntos verticales). Si la aplicación dispone de pestañas (*tabs*), estas podrán situarse en la barra de acciones. También pueden añadirse otros elementos, como listas desplegables y otros tipos de *widgets* incrustados, como el *widget* de búsqueda que veremos más adelante.



Existen dos clases que nos permiten añadir la barra de acciones: `ActionBar` y `ToolBar`. La clase `ActionBar` aparece en la versión 3.0. Por defecto, la barra de acciones es incluida en todas las actividades. Si queremos que no aparezca tenemos que asignar un tema especial a la actividad. Por ejemplo, `Theme.AppCompat.NoActionBar` o cualquiera que acabe en `.NoActionBar`.

La clase `ToolBar` aparece con la versión 5.0. Cambia el diseño de la barra de acciones para que siga las especificaciones de Material Design. Puede usarse en versiones anteriores dado que no se incorpora al API de la versión 5.0, si no a una la librería de compatibilidad `appcompat`. A diferencia de ActionBar, la barra de acciones no es incrustada de forma automática, si no que hay que incluirla en el layout con la etiqueta `<Toolbar>`. Esto nos permite situarla en la posición que queramos y nos da más opciones de configuración.

Añadir un `ToolBar` a la aplicación es muy sencillo. Normalmente no es necesario realizarlo si al crear un proyecto has seleccionado `Basic Activity` o `Scrolling Activity`, dado que en este caso ya se ha añadido un `Toolbar`. Se ha incluido el siguiente ejercicio para los casos en que partes de un proyecto donde se ha incluido un `ActionBar` (seleccionando `Empty Activity`) o en proyectos creados con versiones anteriores.



Ejercicio: *Añadir una barra de acciones con Toolbar.*

1. Crea un nuevo proyecto basado en `Empty Activity` o abre uno que en su layout no aparezca la etiqueta `<Toolbar>`.
2. El primer paso va a ser anular la creación automática de la barra de acciones basada en `ActionBar`. Para eso, accede a *res/values/styles.xml* y cambia el tema de la aplicación por `Theme.AppCompat.Light.NoActionBar`. Verifica que al ejecutar la barra desaparece.
3. También puedes ocultar la barra de estado de Android, donde se muestran las notificaciones y la hora. Para ello, añade el siguiente ítem al tema de la aplicación:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
   <item name="android:windowFullscreen">true</item>
   …
```

4. Añade al layout de la actividad el código siguiente dentro del contenedor raíz:

```
<androidx.appcompat.widget.Toolbar
     android:id="@+id/toolbar"
     android:layout_width="match_parent"
     android:layout_height="?attr/actionBarSize"
     android:background="?attr/colorPrimary"
     android:elevation="4dp"
     android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
     app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
```

Con los dos últimos atributos podemos controlar el tema aplicado al `Toolbar` y al menú de overflow.

5. Si lo añades dentro de un `ConstraintLayout` Asegúrate de indicar las restricciones de posición:

```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
```

6. Verifica que la actividad desciende de `AppCompatActivity`.
7. Añade en el método `onCreate()` el siguiente código:

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```

```
setSupportActionBar(toolbar)
```

8. Has de utilizar el import `androidx.appcompat.widget.Toolbar`.
9. Ejecuta la aplicación. El resultado es equivalente a crear la barra de acciones de forma implícita con `ActionBar`. De momento en la barra de acciones no se mostrará nada.
10. Puedes situar libremente la barra de acciones dentro del layout. Aunque lo correcto es que aparezca en la parte superior.



Ejercicio: *Añadiendo un menú a la barra de acciones*

Podemos asignar un menú a nuestra actividad de forma muy sencilla.

1. Si estás desarrollando el proyecto Asteroides. Pulsa con el botón derecho sobre la carpeta res y selecciona la opción *New > Android resource file*. En el campo *File name:* selecciona *menu_main* y en el campo *Resource type:* selecciona *Menu*.

   Si desarrollas Mis Lugares. Abre el fichero res / *menu* / *menu_main.xml*.

   NOTA: El fichero de menú se crea automáticamente si seleccionas una actividad de tipo: Basic Activity o Scrolling Activity.

2. Reemplaza el contenido que se muestra a continuación:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/ tools"
      tools:context=".MainActivity">
   <item android:id="@+id/action_settings"
         android:title="Configuración"
         android:icon="@android:drawable/ic_menu_preferences"
         android:orderInCategory="5"
         app:showAsAction="never"/>
   <item android:title="Acerca de..."
         android:id="@+id/acercaDe"
```



```
         android:icon="@android:drawable/ic_menu_info_details"
         android:orderInCategory="10"
         app:showAsAction="ifRoom|withText"/>
   <item android:title="Buscar"
         android:id="@+id/menu_buscar"
         android:icon="@android:drawable/ic_menu_search"
         android:orderInCategory="115"
         app:showAsAction="always|collapseActionView"/>
</menu>
```

Como puedes ver cada ítem de menú tiene cinco atributos: `id` que permite identificarlo desde el código; `title`, para asociarle un texto; `icon`, para asociarle un icono; `orderInCategory`, permite ordenar las acciones según el número indicado. Las acciones con un número más pequeño se sitúan más a la izquierda. Si no caben todas las acciones en la barra, las que tienen un número mayor se mueven al menú de *Overflow*. Finalmente, el atributo `showAsAction` permite indicar que acciones son ocultadas en el menú de *Overflow* y cuales están siempre visibles. Si se indica `always` se mostrarán siempre, sin importar si caben o no. El uso de estas acciones debería limitarse, lo ideal es que haya una o dos, ya que al forzar que se visualicen muchas podrían verse incorrectamente. Las acciones que indiquen `ifRoom` se mostrarán en la barra de acciones si hay espacio disponible, y se moverán al menú de *Overflow* si no lo hay. En esta categoría se deberían encontrar la mayoría de las acciones. Si se indica `never`, la acción nunca se mostrará en la barra de acciones, sin importar el espacio disponible. En este grupo se deberían situar acciones como modificar las preferencias, que deben estar disponibles para el usuario, pero no visibles en todo momento.

3. Para activar el menú, has de introducir el siguiente código en la actividad que muestra el menú. Posiblemente solo tengas que incluir el texto subrayado.

```
@Override public boolean onCreateOptionsMenu(Menu menu) {
   getMenuInflater().inflate(R.menu.menu_main, menu);
   return true; /** true -> el menú ya está visible */
}

@Override public boolean onOptionsItemSelected(MenuItem item) {
   int id = item.getItemId();
   if (id == R.id.action_settings) {
      return true;
   }
   if (id == R.id.acercaDe) {
      lanzarAcercaDe(null);
      return true;
   }
   return super.onOptionsItemSelected(item);
}
```

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
  menuInflater.inflate(R.menu.menu_main, menu)
  return true
}
```

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
   return when (item.itemId) {
      R.id.action_settings -> true
      R.id.acercaDe -> {
         lanzarAcercaDe()
         true
      }
      else -> super.onOptionsItemSelected(item)
   }
}
```

4. Ejecuta la aplicación. Podrás ver como aparece la barra de acciones en la parte de arriba, con los botones que hemos definido.



Android Studio incorpora un editor visual de menús que nos permite crear menús sin necesidad de escribir código xml.



3.4. Acceder a objetos globales de la aplicación

Cada uno de los componentes de una aplicación se escribe en una clase separada. Esto hace que en muchas ocasiones resulte complicado compartir objetos entre estos componentes.

Para poder acceder a una información global desde cualquier clase de nuestro proyecto, podemos utilizar el modificador `static`. De esta forma, no será necesario conocer la referencia a un objeto de la clase, solo con indicar el nombre de la clase podremos acceder a esta información.

Otra alternativa, muy similar a la anterior, es utilizar el patrón *Singleton*. Una clase definida con este patrón solo dispondrá de una instancia a la que se podrá acceder desde cualquier sitio utilizando un método estático. Lo veremos más adelante.

Una tercera alternativa específica de Android consiste en crear un descendiente de la clase `Application`. En el siguiente punto se explica cómo hacerlo.

3.4.1. La clase Application

Esta clase ha sido creada en Android para almacenar información global a toda la aplicación.



Vídeo[Tutorial]: *La clase Application en Android*

Veamos cómo usarla en tres pasos:

1. Crea un descendiente de `Applicaction` que contenga la información global y los métodos asociados para acceder a esta información. Mira el ejemplo:

```
public class Aplicacion extends Application {
   private int saldo;
   @Override public void onCreate() {
      super.onCreate();
      SharedPreferences pref = getSharedPreferences("pref", MODE_PRIVATE);
      saldo = pref.getInt("saldo_inicial", -1);
   }

   public int getSaldo(){
      return saldo;
   }

   public void setSaldo(int saldo){
      this.saldo=saldo;
   }
}
```

```
class Aplicacion : Application() {
   var saldo: Int = 0

   override fun onCreate() {
      super.onCreate()
      val pref = getSharedPreferences("pref", MODE_PRIVATE)
      saldo = pref.getInt("saldo_inicial", -1)
   }
}
```

En nuestra aplicación queremos que el usuario disponga de un `saldo` de puntos, con los que podrá ir desbloqueando ciertas características especiales. La clase `Application` es descendiente de `Context`, por lo que tendremos acceso a todos los métodos relativos a nuestro contexto. Entre estos métodos se incluye `getSharedPreferences`, para acceder a un fichero de preferencias almacenado en la memoria interna de nuestra aplicación. La clase `Application` permite sobrescribir los siguientes:

`onCreate()` llamado cuando se cree la aplicación. Puedes usarlo para inicializar los datos.

`onConfigurationChanged(Configuration nuevaConfig)` llamado cuando se realicen cambios en la configuración del dispositivo, mientras que la aplicación se está ejecutando.

`onLowMemory()` llamado cuando el sistema se está quedando sin memoria. Trata de liberar toda la memoria que sea posible.

`onTrimMemory(int nivel)` (*desde nivel API 14*) llamado cuando el sistema determina que es un buen momento para que una aplicación recorte memoria. Esto ocurrirá, por ejemplo, cuando está en el fondo de la pila de actividades y no hay suficiente memoria para mantener tantos procesos en segundo plano. Además, se nos pasa como parámetro el nivel de necesidad. Algunos valores posibles son: `TRIM_MEMORY_COMPLETE`, `TRIM_MEMORY_BACKGROUND`, `TRIM_MEMORY_MODERATE`, ...

2. Registra la clase creada en *AndroidManifest*. Para ello busca la etiqueta `<application>` y añade el atributo `name`, con el nombre de la clase creada:

```
...
<application
      android:name="Aplicacion"
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme">
...
```

3. Puedes obtener una referencia a tu clase `Application` con este código:

```
Aplicacion aplicacion = (Aplicacion) contexto.getApplication();
```

```
val aplicacion = contexto.application as Aplicacion
```

Donde `contexto` es una referencia a la clase `Context`. En caso de estar en un descendiente de esta clase (como `Activity`, `Service`,...) no es necesario disponer de esta referencia, la misma clase ya es un `Context`. Por lo tanto, podríamos escribir:

```
Aplicacion aplicacion = (Aplicacion) getApplication();
```

```
val aplicacion = application as Aplicacion
```

o incluso directamente:

```
int miSaldo = ((Aplicacion) getApplication()).getSaldo();
```

```
val miSaldo = (application as Aplicacion).saldo
```

Ejercicio: *Añadir la clase Application en Mis Lugares*

1. Abre el proyecto Mis Lugares.

2. Vamos a empezar creando una nueva clase. Para ello pulsa con el botón derecho sobre el *java / com.example.mislugares* y selecciona *New > Java Class o New > Kotlin File / Class* . Introduce como nombre de la clase `Aplicacion`. En ella vamos a almacenar un objeto que queremos usar globalmente en toda la aplicación. Reemplaza su código por el siguiente:

```
public class Aplicacion extends Application {

   public RepositorioLugares lugares = new LugaresLista();

   @Override public void onCreate() {
      super.onCreate();
   }
}
```

```
class Aplicacion : Application() {
   val lugares = LugaresLista()
}
```

Nota: Tras incluir nuevas clases tendrás que indicar los imports adecuados. Pulsa «Alt+Intro» en Android Studio para que lo haga automáticamente.

3. Registra el nombre de la clase en *AndroidManifest*, añadiendo la línea que aparece en negrita.

```
<application
      android:name="Aplicacion"
      android:allowBackup="true"
      ...
```

4. En el primer capítulo se creó el proyecto *MisLugaresJava/Kotlin*. Abre este proyecto y selecciona las clases `GeoPunto`, `Lugar`, `RepositorioLugares`, `LugaresLista` y `TipoLugar` (puedes seleccionar varios ficheros manteniendo la tecla *Ctrl* pulsada). Con el botón derecho selecciona la opción *Copy*. Con el botón derecho pulsa sobre *com.example.mislugares* del proyecto *MisLugares* y selecciona la opción *Paste*.

5. Puedes ejecutar el proyecto para verificar que sigue funcionando. No has de notar diferencias con respecto a la versión anterior, no hemos añadido nuevas funcionalidades.

3.4.2. Uso del patrón Singleton <opcional>

Nota: Se trata de una sección avanzada. Puedes saltarte este apartado en una primera lectura.



Vídeo[Tutorial]: *El patrón de diseño Singleton.*

Otra alternativa para almacenar información global a una aplicación es utilizar el patrón *Singleton*. Una clase definida con este patrón solo dispondrá de una instancia,

a la que se podrá acceder desde cualquier clase utilizando un método estático. Una posible implementación de este patrón en Java se muestra a continuación:

```
public class Singleton {
   // Esta será la instancia única de esta clase
   private static Singleton INSTANCIA = new Singleton();

   // El constructor es private para evitar su acceso desde fuera.
   private Singleton() {}

   // Método para obtener la única instancia de la clase
   public static Singleton getInstancia() {
      return INSTANCIA;
   }
}
```

Para obtener la instancia de la clase escribimos desde cualquier sitio:

```
Singleton referencia = Singleton.getInstancia();
```

De hecho, esta es la única forma de acceder a la clase. Al no disponer de constructores públicos no podremos crear nuevos objetos.

Veamos cómo se implementaría el ejemplo anterior, donde se almacenaba el saldo de una aplicación, pero esta vez con el patrón Singleton. El primer problema que se nos plantea es la necesidad de disponer del contexto de la aplicación para poder acceder a las preferencias. Como el *Singleton* no dispone de constructor donde indicar esta información, nos vemos obligados a crear un método para su inicialización (`inicializa()`).

```
public class Saldo {
   private static Saldo INSTANCIA = new Saldo();
   // En Android casi siempre necesitas conocer el contexto
   private Context contexto;
   // Otras variables de la clase
   private int saldo = -1;

   private Saldo() {}

   public static Saldo getInstancia() {
      return INSTANCIA;
   }

   // Método para inicializar el objeto
   public void inicializa(Context contexto){
      this.contexto = contexto;
      SharedPreferences pref = contexto.getSharedPreferences("pref",
               Context.MODE_PRIVATE);
      saldo = pref.getInt("saldo_inicial", -1);
   }

   public int getSaldo() {
      return saldo;
   }
```



```
   public void putSaldo(int saldo) {
      this.saldo = saldo;
   }
}
```

Para utilizar esta clase puedes usar el siguiente código:

```
Saldo saldo = Saldo.getInstancia();
saldo.inicializa(contexto);
int n = saldo.getSaldo();
```

Pero cuidado, asegúrate de llamar a `inicializa()` antes de usarla.

Práctica: *Acceso a información global con el patrón Singleton*

1. Crea una nueva clase con nombre `LugaresSingleton` que siga el patrón *Singleton*. Si lo deseas puedes usar la opción del menú *File/New/Singleton*.
2. Añade como variable privada el objeto `lugares` de tipo `LugaresLista`.
3. Añade el método `inicializa(Context)` y el método *getter*.
4. En todos los ejercicios sobre Mis Lugares, accede al objeto `lugares` usando el singleton en lugar de la clase `Application`.

Aunque la clase `Application` fue creada para almacenar información global a la aplicación, desde la misma documentación de Android se nos recomienda utilizar el patrón *Singleton* para este propósito:

> *"There is normally no need to subclass Application. In most situations, static singletons can provide the same functionality in a more modular way"*

Es decir, la clase `Saldo`, basada en patrón *Singleton*, tiene una mayor modularidad que la clase `Aplicacion`, descendiente de `Application`, dado que puede ser reutilizada en otros proyectos sin ser modificada. Aunque, el uso de `Application` también tendría sus ventajas: un código más limpio y que no requiere ser inicializado desde fuera de la clase. Dejamos en manos del lector la decisión de utilizar uno u otro mecanismo.

Nota: *El patrón Singleton tiene sus detractores dado que presentan problemas a la hora de testear las aplicaciones.*



Preguntas de repaso: *Application y Singleton*

3.5. Uso de la arquitectura Clean en Mis Lugares

Este texto trata de dar una visión introductoria a la programación en Android, por lo tanto, el diseño de arquitecturas de software queda algo alejado de sus objetivos. No obstante, pensamos que puede ser interesante comentar algunos conceptos y tratar de seguir unos ejemplos con una arquitectura adecuada.

A medida que una aplicación crece comprobarás que cada vez resulta más difícil de mantener. Si no seguimos unas reglas claras en el desarrollo, el caos está garantizado. Por ejemplo, un error típico en Android suele ser dar demasiadas responsabilidades a las actividades. `MainActivity` puede llegar a tener cientos, o incluso miles, de líneas de código. Si estas responsabilidades las separamos en varias clases, el código será más fácil de entender, tendrá menos errores y será más reutilizable.

Para estructurar las clases de la aplicación nos vamos a inspirar en la arquitectura Clean[20]. Aunque de forma muy simplificada, no vamos a realizar inyección de dependencias, usar patrones como modelo-vista-controlador, ni otros temas que complicarían en exceso la aplicación. En una primera aproximación y dado el tamaño de la aplicación, posiblemente sería excesivo aplicar estas técnicas.

Clean no es una arquitectura tal cual, si no una serie de guías y buenas prácticas en el desarrollo de software. Fue definida por Rober C Martin (*Uncle* Bob) en su charla "*Architecture the lost years*", donde exponía una serie de problemas y el alto acoplamiento de los desarrollos de software tanto a los modelos de datos como a la interfaz.

Clean define una serie de capas y otorga una responsabilidad a cada una, pero no entra en profundidad en los detalles de implementación y cómo se deben resolver los problemas.

El objetivo es escribir software que esté lo menos acoplado posible a nuestro modelo de datos, a la representación de este y al framework que estemos usando. Esto va a incrementar la estabilidad de nuestro código ya que va a hacer más fácil cambiar las partes dependientes del sistema. Va a facilitar la portabilidad a otros entornos (como iOS o Web) dado que gran parte del código es independiente del framework. También va a permitir postergar decisiones de implementación, como por ejemplo la persistencia o el uso de red. Podemos hacer una primera versión de nuestro software que guarde los datos de forma local y de una forma sencilla cambiarlo a online. O elegir el framework de persistencia cuando nos sea necesario y no antes.

Aunque dentro de Clean existe variantes, en la aplicación Mis Lugares vamos a organizar las clases en 4 capas:

Capa de Modelo

También se utiliza el nombre de Dominio o Lógica de Negocio. Está formada por las clases que representan la lógica interna de la aplicación y cómo representamos los datos con los que vamos a trabajar. Muchas clases de esta capa se conocen como

[20] https://devexperto.com/clean-architecture-android/

POJO (*Plain Old Java Object*) al tratarse de clases Java puras. Ejemplos de POJO serían las clases `Lugar` o `GeoPunto`. No es conveniente que en estas clases se utilicen APIs externos. Si abres las clases que hemos indicado, podrás comprobar que no necesitan ningún `import`.

Capa de Datos

En esta capa estarían las clases encargadas de guardar de forma permanente los datos y cómo acceder a ellos. Suelen representar bases de datos, servicios Web, preferencias, ficheros JSON... También es conocida como capa de Almacenamiento o Persistencia.

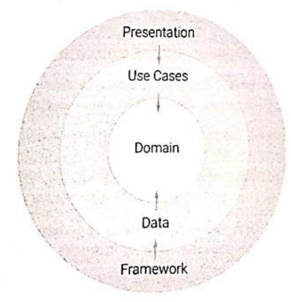
Capa de Casos de Uso

Los casos de uso son clases que van a definir las operaciones que el usuario puede realizar con nuestra aplicación. Esta capa no sería estrictamente necesaria (por ejemplo, en Asteroides no la vamos a utilizar), pero resulta muy interesante para tener enumeradas las diferentes acciones que vamos a implementar. Además, va a permitir quitar mucha responsabilidad a las actividades. Los casos de uso también se conocen como *interactors*.

Capa de Presentación

Representa la interfaz de usuario, por lo que está formada por las actividades, fragments, vistas y otros elementos con los que interactúa el usuario.

Una de las características más importantes de esta arquitectura es la **regla de dependencia entre capas**. Para representar las dependencias se suelen usar un diagrama en forma de círculos concéntricos. Las capas más internas son aquellas que están más cercanas a nuestra lógica de dominio, no deben depender de las capas más externas del software, aquellas que están más cerca a los agentes externos como el framework, o el interfaz de usuario.



Por ejemplo, la clase `Lugar` que pertenecería a la capa de Modelo, va a poder ser utilizada por el resto de las capas y no puede usar otras clases que no sean de su capa. Por el contrario, una actividad perteneciente a la capa de presentación no debería usarse por el resto de las capas y puede usar cualquier capa interior.

3.2. Comunicación entre actividades

Cuando una actividad ha de lanzar a otra actividad, en muchos casos necesita enviarle cierta información.



Vídeo[tutorial]: *Intercambio de datos entre actividades*

Android nos permite este intercambio de datos utilizando el mecanismo que se describe a continuación:

Cuando lances una actividad B, desde la actividad A, usa el siguiente código en A:

```
Intent intent = new Intent(this, MI_CLASE.class);
intent.putExtra("usuario", "Pepito Perez");
intent.putExtra("edad", 27);
startActivity(intent);
```

```
val intent = Intent(this, MI_CLASE::class.java)
intent.putExtra("usuario", "Pepito Perez")
intent.putExtra("edad", 27)
startActivity(intent)
```

En la actividad lanzada (B) podemos recoger los datos de la siguiente forma:

```
Bundle extras = getIntent().getExtras();
String s = extras.getString("usuario");
int i = extras.getInt("edad");
```

```
val extras = intent.extras
val s = extras?.getString("usuario")?:"sin usuario"
val i = extras?.getInt("edad")?:-1
```

Cuando la actividad lanzada (B) termina, si lo desea, podrá enviar datos de vuelta. Para ello añade en la actividad B el siguiente código:

```
Intent intent = new Intent();
intent.putExtra("resultado","valor");
setResult(RESULT_OK, intent);
finish();
```

```
val intent = Intent()
intent.putExtra("resultado", "valor")
setResult(Activity.RESULT_OK, intent)
finish()
```

Es posible que el trabajo realizado en la actividad B sea cancelado. Para este caso añade:

```
Intent intent = new Intent();
setResult(RESULT_CANCEL, intent);
finish();
```



```
val intent = Intent()
setResult(Activity.RESULT_CANCEL, intent)
finish()
```

En la actividad que hizo la llamada (A) has de poder recoger estos datos. Para ello tendremos que lanzar la actividad con `startActivityForResult(Intent, int)` en lugar de `startActivityForResult(Intent)`, donde el segundo parámetro es un entero con un código que identifica a la actividad que lanzamos. Además, tendremos que sobrescribir el método `onActivityResult(int, int, Intent)`, donde en el primer parámetro se devuelve el mismo código que indicamos cuando hicimos la llamada. El segundo, si el resultado ha sido ok o cancelado. Y el tercero, un `Intent` donde se incluyen las variables devueltas:

```
Intent intent = new Intent(this, MI_CLASE.class);
startActivityForResult(intent, 1234);

@Override protected void onActivityResult(int requestCode, int resultCode,
                                          Intent data){
   super.onActivityResult(requestCode, resultCode, data);
   if (requestCode==1234 && resultCode==RESULT_OK) {
      String res = data.getExtras().getString("resultado");
   }
}
```

```
val intent = Intent(this, MI_CLASE::class.java)
startActivityForResult(intent, 1234)

override fun onActivityResult(requestCode: Int, resultCode: Int,
                              data: Intent?) {
   if (requestCode == 1234 && resultCode == Activity.RESULT_OK) {
      val res = data?.extras?.getString("resultado")?:"sin resultado"
   }
}
```

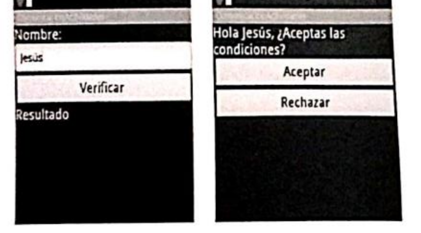
Desde la actividad A se podrían llamar a varias actividades. Sin embargo, solo podemos tener un método `onActivityResult()`. Por esta razón, resulta necesario identificar cada actividad lanzada con un código. Así podremos diferenciar entre los distintos datos devueltos.



Práctica: *Comunicación entre actividades*

1. Crea un nuevo proyecto con nombre *Comunicación Actividades* y tipo *Empty Activity*.
2. El *layout* de la actividad inicial ha de ser similar al que se muestra abajo a la izquierda.
3. Introduce el código para que cuando se pulse el botón *Verificar* se arranque una segunda actividad. A esta actividad se le pasará como parámetro el nombre introducido en el `EditText`.

4. El *layout* correspondiente a la segunda *actividad se muestra a la derecha.*
5. Al arrancar la actividad, el texto del primer `TextView` ha de modificarse para que ponga "Hola +nombre recibido+, ¿Aceptas las condiciones?"
6. En esta actividad se podrán pulsar dos botones, de forma que se devuelva a la actividad principal el `String` «Aceptado», al pulsar en Aceptar. Al pulsar el botón Cancelar también se regresará a la actividad anterior.
7. En la actividad principal se modificará el texto del último `TextView` para que ponga «Resultado: Aceptado» o «Resultado: Rechazado», según lo recibido.





Preguntas de repaso: *Comunicación entre Actividades*

3.3. La barra de acciones (*Toolbar*)



Vídeo[tutorial]: *La barra de acciones (*Toolbar*)*

Desde la versión 3.0, se introdujo en Android un nuevo elemento en la interfaz de usuario: la barra de acciones. Esta se sitúa en la parte superior de la pantalla, fue creada para que el usuario tuviera una experiencia unificada a través de las distintas aplicaciones. La barra de acciones aglutina varios elementos; los más habituales son el nombre de la aplicación, el botón para abrir el *Navigation Drawer* y los botones de acciones frecuentes. Las acciones menos utilizadas se sitúan en un menú desplegable, que se abrirá desde el botón *Overflow* (se representa con tres puntos verticales). Si la aplicación dispone de pestañas (*tabs*), estas podrán situarse en la barra de acciones. También pueden añadirse otros elementos, como listas desplegables y otros tipos de *widgets* incrustados, como el *widget* de búsqueda que veremos más adelante.



Existen dos clases que nos permiten añadir la barra de acciones: `ActionBar` y `ToolBar`. La clase `ActionBar` aparece en la versión 3.0. Por defecto, la barra de acciones es incluida en todas las actividades. Si queremos que no aparezca tenemos que asignar un tema especial a la actividad. Por ejemplo. `Theme.AppCompat.NoActionBar` o cualquiera que acabe en `.NoActionBar`.

La clase `ToolBar` aparece con la versión 5.0. Cambia el diseño de la barra de acciones para que siga las especificaciones de Material Design. Puede usarse en versiones anteriores dado que no se incorpora al API de la versión 5.0, si no a una la librería de compatibilidad `appcompat`. A diferencia de ActionBar, la barra de acciones no es incrustada de forma automática, si no que hay que incluirla en el layout con la etiqueta `<Toolbar>`. Esto nos permite situarla en la posición que queramos y nos da más opciones de configuración.

Añadir un `ToolBar` a la aplicación es muy sencillo. Normalmente no es necesario realizarlo si al crear un proyecto has seleccionado `Basic Activity` o `Scrolling Activity`, dado que en este caso ya se ha añadido un `Toolbar`. Se ha incluido el siguiente ejercicio para los casos en que partes de un proyecto donde se ha incluido un `ActionBar` (seleccionando `Empty Activity`) o en proyectos creados con versiones anteriores.



Ejercicio: *Añadir una barra de acciones con Toolbar.*

1. Crea un nuevo proyecto basado en `Empty Activity` o abre uno que en su layout no aparezca la etiqueta `<Toolbar>`.
2. El primer paso va a ser anular la creación automática de la barra de acciones basada en `ActionBar`. Para eso, accede a *res/values/styles.xml* y cambia el tema de la aplicación por `Theme.AppCompat.Ligth.NoActionBar`. Verifica que al ejecutar la barra desaparece.
3. También puedes ocultar la barra de estado de Android, donde se muestran las notificaciones y la hora. Para ello, añade el siguiente ítem al tema de la aplicación:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
    <item name="android:windowFullscreen">true</item>
    …
```

4. Añade al layout de la actividad el código siguiente dentro del contenedor raíz:

```
<androidx.appcompat.widget.Toolbar
    android:id="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="?attr/actionBarSize"
    android:background="?attr/colorPrimary"
    android:elevation="4dp"
    android:theme="@style/ThemeOverlay.AppCompat.ActionBar"
    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>
```

Con los dos últimos atributos podemos controlar el tema aplicado al `Toolbar` y al menú de *overflow*.

5. Si lo añades dentro de un `ConstraintLayout` Asegúrate de indicar las restricciones de posición:

```
app:layout_constraintTop_toTopOf="parent"
app:layout_constraintLeft_toLeftOf="parent"
```

6. Verifica que la actividad desciende de `AppCompatActivity`.
7. Añade en el método `onCreate()` el siguiente código:

```
Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
setSupportActionBar(toolbar);
```

```
setSupportActionBar(toolbar)
```

8. Has de utilizar el import `androidx.appcompat.widget.Toolbar`.
9. Ejecuta la aplicación. El resultado es equivalente a crear la barra de acciones de forma implícita con `ActionBar`. De momento en la barra de acciones no se mostrará nada.
10. Puedes situar libremente la barra de acciones dentro del layout. Aunque lo correcto es que aparezca en la parte superior.



Ejercicio: *Añadiendo un menú a la barra de acciones*

Podemos asignar un menú a nuestra actividad de forma muy sencilla.

1. Si estás desarrollando el proyecto Asteroides. Pulsa con el botón derecho sobre la carpeta *res* y selecciona la opción *New > Android resource file*. En el campo *File name:* selecciona *menu_main* y en el campo *Resource type:* selecciona *Menu*.

   Si desarrollas Mis Lugares. Abre el fichero *res / menu / menu_main.xml*.

   NOTA: El fichero de menú se crea automáticamente si seleccionas una actividad de tipo: Basic Activity o Scrolling Activity.

2. Reemplaza el contenido que se muestra a continuación:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:tools="http://schemas.android.com/ tools"
      tools:context=".MainActivity">
   <item android:id="@+id/action_settings"
         android:title="Configuración"
         android:icon="@android:drawable/ic_menu_preferences"
         android:orderInCategory="5"
         app:showAsAction="never"/>
   <item android:title="Acerca de..."
         android:id="@+id/acercaDe"
         android:icon="@android:drawable/ic_menu_info_details"
         android:orderInCategory="10"
         app:showAsAction="ifRoom|withText"/>
   <item android:title="Buscar"
         android:id="@+id/menu_buscar"
         android:icon="@android:drawable/ic_menu_search"
         android:orderInCategory="115"
         app:showAsAction="always|collapseActionView"/>
</menu>
```

Como puedes ver cada ítem de menú tiene cinco atributos: `id` que permite identificarlo desde el código; `title`, para asociarle un texto; `icon`, para asociarle un icono; `orderInCategory`, permite ordenar las acciones según el número indicado. Las acciones con un número más pequeño se sitúan más a la izquierda. Si no caben todas las acciones en la barra, las que tienen un número mayor se mueven al menú de *Overflow*. Finalmente, el atributo `showAsAction` permite indicar que acciones son ocultadas en el menú de *Overflow* y cuales están siempre visibles. Si se indica `always` se mostrarán siempre, sin importar si caben o no. El uso de estas acciones debería limitarse, lo ideal es que haya una o dos, ya que al forzar que se visualicen muchas podrían verse incorrectamente. Las acciones que indiquen `ifRoom` se mostrarán en la barra de acciones si hay espacio disponible, y se moverán al menú de *Overflow* si no lo hay. En esta categoría se deberían encontrar la mayoría de las acciones. Si se indica `never`, la acción nunca se mostrará en la barra de acciones, sin importar el espacio disponible. En este grupo se deberían situar acciones como modificar las preferencias, que deben estar disponibles para el usuario, pero no visibles en todo momento.

3. Para activar el menú, has de introducir el siguiente código en la actividad que muestra el menú. Posiblemente solo tengas que incluir el texto subrayado.

```
@Override public boolean onCreateOptionsMenu(Menu menu) {
   getMenuInflater().inflate(R.menu.menu_main, menu);
   return true; /** true -> el menú ya está visible */
}

@Override public boolean onOptionsItemSelected(MenuItem item) {
   int id = item.getItemId();
   if (id == R.id.action_settings) {
      return true;
   }
   if (id == R.id.acercaDe) {
      lanzarAcercaDe(null);
      return true;
   }
   return super.onOptionsItemSelected(item);
}
```

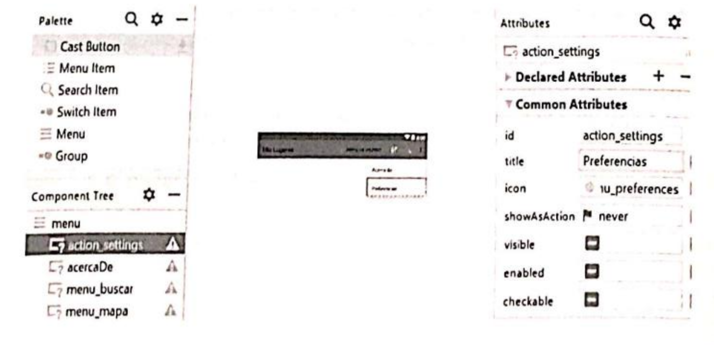
```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
   menuInflater.inflate(R.menu.menu_main, menu)
   return true
}
```

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
   return when (item.itemId) {
      R.id.action_settings -> true
      R.id.acercaDe -> {
         lanzarAcercaDe()
         true
      }
      else -> super.onOptionsItemSelected(item)
   }
}
```

4. Ejecuta la aplicación. Podrás ver como aparece la barra de acciones en la parte de arriba, con los botones que hemos definido.



Android Studio incorpora un editor visual de menús que nos permite crear menús sin necesidad de escribir código xml.



3.4. Acceder a objetos globales de la aplicación

Cada uno de los componentes de una aplicación se escribe en una clase separada. Esto hace que en muchas ocasiones resulte complicado compartir objetos entre estos componentes.

Para poder acceder a una información global desde cualquier clase de nuestro proyecto, podemos utilizar el modificador `static`. De esta forma, no será necesario conocer la referencia a un objeto de la clase, solo con indicar el nombre de la clase podremos acceder a esta información.

Otra alternativa, muy similar a la anterior, es utilizar el patrón *Singleton*. Una clase definida con este patrón solo dispondrá de una instancia a la que se podrá acceder desde cualquier sitio utilizando un método estático. Lo veremos más adelante.

Una tercera alternativa específica de Android consiste en crear un descendiente de la clase `Application`. En el siguiente punto se explica cómo hacerlo.

3.4.1. La clase Application

Esta clase ha sido creada en Android para almacenar información global a toda la aplicación.



Vídeo[Tutorial]: *La clase Application en Android*

Veamos cómo usarla en tres pasos:

1. Crea un descendiente de `Applicaction` que contenga la información global y los métodos asociados para acceder a esta información. Mira el ejemplo:

```
public class Aplicacion extends Application {
   private int saldo;
   @Override public void onCreate() {
      super.onCreate();
      SharedPreferences pref = getSharedPreferences("pref", MODE_PRIVATE);
      saldo = pref.getInt("saldo_inicial", -1);
   }

   public int getSaldo(){
      return saldo;
   }

   public void setSaldo(int saldo){
      this.saldo=saldo;
   }
}
```

```
class Aplicacion : Application() {
   var saldo: Int = 0

   override fun onCreate() {
      super.onCreate()
      val pref = getSharedPreferences("pref", MODE_PRIVATE)
      saldo = pref.getInt("saldo_inicial", -1)
   }
}
```

En nuestra aplicación queremos que el usuario disponga de un `saldo` de puntos, con los que podrá ir desbloqueando ciertas características especiales. La clase `Application` es descendiente de `Context`, por lo que tendremos acceso a todos los métodos relativos a nuestro contexto. Entre estos métodos se incluye `getSharedPreferences`, para acceder a un fichero de preferencias almacenado en la memoria interna de nuestra aplicación. La clase `Application` permite sobrescribir los siguientes:

`onCreate()` llamado cuando se cree la aplicación. Puedes usarlo para inicializar los datos.

Nota: En la literatura la capa de casos de uso es más interna que la de datos. En esta implementación se ha realizado al revés. Reamente no vamos a seguir la Arquitectura Clean de forma estricta. En nuestra implementación la capa de Usos de Datos va a tener dependencias con la capa de Presentación, cosa que habría que evitar.

Organizando las clases en paquetes

Cuando trabajas con muchos ficheros suele ser muy práctico organizarlos en carpetas. Podrías tener todos los ficheros de una asignatura en una misma carpeta, pero seguramente prefieras crear una carpeta para cada unidad o una carpeta por tipo de fichero (transparencias, ejercicios...).

- com.example.mislugares
  - casos_uso
    - CasosUsoActividades
    - CasosUsoLocalizacion
    - CasosUsoLugar
  - datos
    - LugaresBD
    - LugaresLista
    - RepositorioLugares
  - modelo
    - GeoPunto
    - Lugar
    - TipoLugar
  - presentacion
  - Aplicacion

El número de clases de un proyecto Android también puede ser muy elevado, por lo que resulta complicado localizarlas. Para resolver este problema, resulta frecuente organizar las clases en diferentes paquetes. Podemos usar diferentes criterios, por ejemplo, por módulos del proyecto (como autentificación, visualización, mapas...) Otro criterio podría ser por entidades (como lugares, usuarios...) También podemos utilizar la función de la clase (como actividades, fragments, adaptadores...). En este ejercicio utilizaremos como criterio la capa de la arquitectura (en concreto modelo, datos, casos de uso y presentación). La organización propuesta se muestra a la derecha. Pero eres libre de usar nombres en inglés o definir tu propio criterio. El paquete donde se encuentra cada clase no afectará a los ejercicios.

Ejercicio: *Organizar las clases en paquetes*

1. Pulsa con el botón derecho sobre `com.example.mislugares` y selecciona *New / Package*. Escribe `presentacion`, se creará `com.example.mislugares.presentacion`.
2. Arrastra todas las clases terminadas en `Activity` a este paquete. El proceso de refactorización se realiza de forma bastante automática. Aunque en algunos casos tendrás que realizar algún pequeño ajuste, como hacer público algún método. Añade también en este paquete fragments y adaptadores.
3. Repite este proceso para los paquetes `datos`, `modelo` y `casos_uso`. Para ver dónde se sitúa cada clase puedes usar la imagen anterior.



Ejercicio: *Casos de Uso para lugares*

1. Crea la clase `CasosUsoLugar` dentro del paquete `casos_uso` con el código:


```
public class CasosUsoLugar {
   private Activity actividad;
   private RepositorioLugares lugares;

   public CasosUsoLugar(Activity actividad, RepositorioLugares lugares) {
      this.actividad = actividad;
      this.lugares = lugares;
   }

// OPERACIONES BÁSICAS
   public void mostrar(int pos) {
            Intent i = new Intent(actividad, VistaLugarActivity.class);
            i.putExtra("pos", pos);
            actividad.startActivity(i);
   }
}
```

```
class CasosUsoLugar(val actividad: Activity,
                    val lugares: RepositorioLugares) {
// OPERACIONES BÁSICAS
   fun mostrar(pos: Int) {
      val i = Intent(actividad, VistaLugarActivity::class.java)
      i.putExtra("pos", pos);
      actividad.startActivity(i);
   }
}
```

Dentro de esta clase vamos a añadir diferentes funciones que ejecutarán distintos casos de uso referentes a un lugar. Por ejemplo, compartir un lugar, borrarlo, ... De momento solo añadimos un caso de uso, `mostar(pos)`, que arrancará una actividad mostrando la información del lugar según su posición en el `RecyclerView`.

Se han añadido dos propiedades a la clase. La primera, `actividad`, nos va a permitir extraer el contexto o lanzar otras actividades. ***Nota:** No se cumple la regla de dependencias de la arquitectura Clean, se añade porque va a simplificar el código.* La segunda, `lugares`, nos va a permitir acceder al repositorio de los lugares.

2. Añade en `MainActivity` las siguientes propiedades:

```
private RepositorioLugares lugares;
private CasosUsoLugar usoLugar;

@Override protected void onCreate(Bundle savedInstanceState) {
   …
   lugares = ((Aplicacion) getApplication()).lugares;
   usoLugar = new CasosUsoLugar(this, lugares);
}
```

```
val lugares by lazy { (application as Aplicacion).lugares }
val usoLugar by lazy { CasosUsoLugar(this, lugares) }
```

3. Cada vez que queramos ejecutar este caso de uso usaremos el código:

```
usoLugar.mostrar(pos)
```

Este código será usado en el en siguiente ejercicio. Para evitar que de error puedes comentar el contenido de la función `mostrar()`.



Práctica: *Casos de Uso para arrancar actividades*

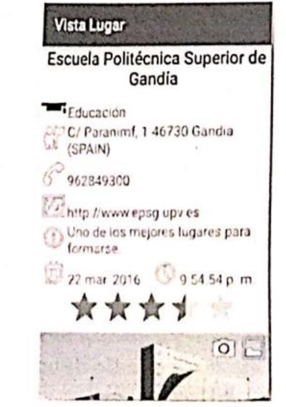
1. Crea la clase `CasosUsoActividades` dentro del paquete `casos_uso`.
2. Crea la función `lanzarAcerdaDe()`. Ha de contener el código necesario para arrancar `AcerdaDeActivity`.
3. Añade en `MainActivity` el código necesario para usar este caso de uso.
4. En esta clase también se puedes añadir otros casos de uso como `lanzarPreferencias()` o `lanzarMapa()`, para abrir las actividades adecuadas.

Una vez que completes la aplicación Mis Lugares, las clases para casos de uso acaban conteniendo decenas de métodos. De esta forma, va a ser muy sencillo saber qué hace cada método, donde está y las dependencias que necesita. De lo contrario, todo este código acabaría en las actividades, que contendría cientos de líneas de código, siendo muy difíciles de mantener.

3.6. Creando actividades en Mis Lugares

3.6.1. Creando la actividad VistaLugarActivity

La actividad `VistaLugarActivity` nos mostrará la información que hemos almacenado de un determinado lugar y nos permitirá realizar una gran cantidad de acciones sobre ese lugar (mostrar en mapa, llamar por teléfono, compartir en redes sociales, etc.). Desde esta actividad podremos cambiar algunos valores de modificación frecuente. En concreto: la valoración, la fecha de visita y la fotografía.





Ejercicio: *Creación de la actividad VistaLugarActivity*

1. Abre el proyecto MisLugares.
2. Descarga http://www.androidcurso.com/images/dcomg/ficheros/mislugares.zip y descomprime en una carpeta. Copia los gráficos que encontrarás en el portapapeles y pégalos dentro de *res/drawable* en el explorador del proyecto.
3. Crea un nuevo *layout* y llámalo *vista_lugar.xml*. Copia el siguiente código para usarlo como base:

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/scrollView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical" >
        <TextView
            android:id="@+id/nombre"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center_vertical"
            android:gravity="center"
            android:text="Nombres del lugar"
            android:textAppearance="?android:attr/textAppearanceLarge" />
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal" >
            <ImageView
                android:id="@+id/logo_tipo"
                android:layout_width="40dp"
                android:layout_height="40dp"
                android:contentDescription="logo del tipo"
                android:src="@drawable/otros" />
            <TextView
                android:id="@+id/tipo"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_gravity="bottom"
                android:textAppearance="?android:attr/textAppearanceMedium"
                android:text="tipo del lugar" />
        </LinearLayout>
    …
        <RatingBar
            android:id="@+id/valoracion"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:layout_gravity="center_horizontal"
            android:rating="3" />
        <FrameLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content" >
            <ImageView
                android:id="@+id/foto"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:adjustViewBounds="true"
                android:contentDescription="fotografía"
                android:src="@drawable/foto_epsg" />
            <LinearLayout
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="right" >
            <ImageView
                android:id="@+id/camara"
                android:layout_width="40dp"
                android:layout_height="40dp"
                android:contentDescription="logo cámara"
                android:src="@android:drawable/ic_menu_camera" />
            <ImageView
                android:id="@+id/galeria"
                android:layout_width="40dp"
                android:layout_height="40dp"
                android:contentDescription="logo galería"
                android:src="@android:drawable/ic_menu_gallery" />
            </LinearLayout>
        </FrameLayout>
    </LinearLayout>
</ScrollView>
```

Observa como el elemento exterior es un `ScrollView`. Esto es conveniente cuando pensamos que los elementos de *layout* no cabrán en la pantalla. En este caso, el usuario podrá desplazar verticalmente el *layout* arrastrando con el dedo. Dado que algunas pantallas pueden ser muy pequeñas, la mayoría de los diseños han de incorporar un `ScrollView`.

Dentro de este elemento tenemos un `LinearLayout` para organizar las vistas verticalmente. La primera vista es un `TextView` cuyo *id* es `nombre`. Se ha asignado un valor para `text` inicial, que será reemplazado por el nombre del lugar. La única función que tiene este texto inicial es ayudarnos en el diseño. El siguiente elemento es un `LinearLayout` vertical que contiene un `ImageView` y un `TextView`. Este elemento se utilizará para indicar el tipo de lugar.

Los puntos suspensivos indican el lugar donde tendrás que insertar el resto de los elementos que no se han incluido (dirección, teléfono, etc.). El siguiente elemento que se incluye es un `RatingBar`, donde podremos introducir una valoración del lugar. El último elemento es un `FrameLayout`, que permite superponer varias vistas. Se dibujarán en el orden en que las indicamos. En el fondo se dibuja un `ImageView` con una fotografía de la EPSG. El atributo `adjustViewBounds` indica que la imagen

sea escalada para que ocupe todo el espacio disponible. Sobre la fotografía se dibujará un `LinearLayout` con dos `ImageView`. Estos botones permitirán cambiar la fotografía desde la cámara o desde la galería.

4. Reemplaza los puntos suspensivos por los elementos que faltan para obtener la apariencia mostrada al principio de este punto. Utiliza los recursos del sistema mostrados en la siguiente tabla. Identifica cada `TextView` con el *id* que se indica.

Recurso para `ImageView`	*Id* para `TextView`
@android:drawable/ic_menu_myplaces	@+id/direccion
@android:drawable/ic_menu_call	@+id/telefono
@android:drawable/ic_menu_mapmode	@+id/url
@android:drawable/ic_menu_info_details	@+id/comentario
@android:drawable/ic_menu_my_calendar	@+id/fecha
@android:drawable/ic_menu_recent_history	@+id/hora

5. Crea la clase `VistaLugarActivity` y reemplaza el código por el siguiente:

```
public class VistaLugarActivity extends AppCompatActivity {
   private RepositorioLugares lugares;
   private CasosUsoLugar usoLugar;
   private int pos;
   private Lugar lugar;
   @Override protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.vista_lugar);
      Bundle extras = getIntent().getExtras();
      pos = extras.getInt("pos", 0);
      lugares = ((Aplicacion) getApplication()).lugares;
      usoLugar = new CasosUsoLugar(this, lugares);
      lugar = lugares.elemento(pos);
      actualizaVistas();
   }
   public void actualizaVistas() {
      TextView nombre = findViewById(R.id.nombre);
      nombre.setText(lugar.getNombre());
      ImageView logo_tipo = findViewById(R.id.logo_tipo);
      logo_tipo.setImageResource(lugar.getTipo().getRecurso());
      TextView tipo = findViewById(R.id.tipo);
      tipo.setText(lugar.getTipo().getTexto());
      TextView direccion = findViewById(R.id.direccion);
      direccion.setText(lugar.getDireccion());
      TextView telefono = findViewById(R.id.telefono);
      telefono.setText(Integer.toString(lugar.getTelefono()));
      TextView url = findViewById(R.id.url);
      url.setText(lugar.getUrl());
      TextView comentario = findViewById(R.id.comentario);
      comentario.setText(lugar.getComentario());
      TextView fecha = findViewById(R.id.fecha);
      fecha.setText(DateFormat.getDateInstance().format(
            new Date(lugar.getFecha())));
```

```
        TextView hora = findViewById(R.id.hora);
        hora.setText(DateFormat.getTimeInstance().format(
                new Date(lugar.getFecha())));
        RatingBar valoracion = findViewById(R.id.valoracion);
        valoracion.setRating(lugar.getValoracion());
        valoracion.setOnRatingBarChangeListener(
                new OnRatingBarChangeListener() {
                    @Override public void onRatingChanged(RatingBar ratingBar,
                                                float valor, boolean fromUser) {
                        lugar.setValoracion(valor);
                    }
        });
    }
}
```

```
class VistaLugarActivity : AppCompatActivity() {
   val lugares by lazy { (application as Aplicacion).lugares }
   val usoLugar by lazy { CasosUsoLugar(this, lugares) }
   var pos = 0
   lateinit var lugar: Lugar

   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.vista_lugar)
      pos = intent.extras?.getInt("pos", 0) ?: 0
      lugar = lugares.elemento(pos)
      actualizaVistas()
   }

   fun actualizaVistas(){
      nombre.text = lugar.nombre
      logo_tipo.imageResource = lugar.tipoLugar.recurso
      tipo.text = lugar.tipoLugar.texto
      direccion.text = lugar.direccion
      telefono.text = Integer.toString(lugar.telefono)
      url.text = lugar.url
      comentario.text = lugar.comentarios
      fecha.text = DateFormat.getDateInstance().format(Date(lugar.fecha))
      hora.text = DateFormat.getTimeInstance().format(Date(lugar.fecha))
      valoracion.rating = lugar.valoracion
      valoracion.setOnRatingBarChangeListener {
         ratingBar, valor, fromUser -> lugar.valoracion = valor
      }
   }
}
```

Nota sobre Java/Kotlin: *Pulsa **Alt-Intro** para que automáticamente se añadan los* imports *con los paquetes que faltan. Dos clases aparecen en varios paquetes, selecciona* java.text.DateFormat *y* java.util.Date.

Se definen tres variables globales para que se pueda acceder a ellas desde cualquier método de la clase: `lugares` corresponde con el repositorio de lugares, cuya referencia se obtiene desde `Application`; `pos` es la posición del elemento que vamos a visualizar; y `lugar` es el elemento en sí.

El método `onCreate()` se ejecutará cuando se cree la actividad y en él tenemos que asociar un *layout* (`setContentView(R.layout.vista_lugar)`) e inicializar todos sus valores. A continuación, se averigua la posición del lugar a mostrar, que ha sido pasado en un *extra*. A partir de la posición obtenemos el objeto `Lugar` a mostrar.

En Java observa cómo se obtiene un objeto de cada uno de los elementos de la vista utilizando el método `findViewById()`. **En Kotlin** esta operación se realiza de forma automática. A continuación, este objeto se modifica según el valor del lugar que estamos representando. Al final se realiza una acción especial con el objeto `valoracion`, utilizando el método `setOnRatingBarChangeListener()` para asignarle un escuchador de eventos al `RatingBar` que se crea allí mismo. Este escuchador de evento se activará cuando el usuario modifique la valoración. El código a ejecutar consiste en modificar la propiedad `valoracion` del objeto `lugar` con la nueva valoración.

6. Abre la clase `TipoLugar` y asigna un recurso *drawable* a cada tipo de lugar. Puedes utilizar la opción de autocompletar, es decir, escribe `R.drawable.` y espera a que el sistema te dé una alternativa.

```
public enum TipoLugar {
   OTROS ("Otros", R.drawable.otros),
   RESTAURANTE ("Restaurante", R.drawable.restaurante),
   BAR ("Bar", R.drawable.bar),
   …
```

7. Añade en `MainActivity` el siguiente método:

```
public void lanzarVistaLugar(View view){
   usoLugar.mostrar(0);
}
```

```
fun lanzarVistaLugar(view: View? = null) {
   usoLugar.mostrar(0)
}
```

Este método lanzará la actividad `VistaLugarActivity` pasándole como posición del lugar a visualizar siempre 0. Más adelante mostraremos algunas alternativas para que el usuario pueda seleccionar el lugar a mostrar.

8. En el método `onOptionsItemSelected()` de la actividad `MainActivity` añade el siguiente código:

```
@Override public boolean onOptionsItemSelected(MenuItem item) {
   int id = item.getItemId();
   …
   if (id == R.id.menu_buscar) {
      lanzarVistaLugar(null);
      return true;
   }
```

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
   return when (item.itemId) {
      …
      R.id.menu_buscar -> {
         lanzarVistaLugar()
         true;
      }
```

9. Ejecuta la aplicación. Aparecerá un error cuando selecciones *Buscar*. Siempre que aparezca un error en ejecución, es el momento de visualizar el *LogCat*. No es sencillo analizar la información que se muestra, pero es muy importante que te acostumbres a buscar la causa del problema en el *LogCat*. En este caso, la información clave se muestra a continuación:

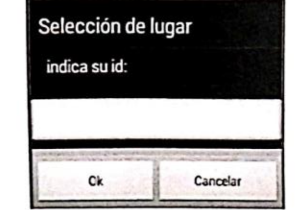
```
LogCat
Search for messages. Accepts Java regexes. Prefix with pid:, app:, t   verbose
Text
Caused by: android.content.ActivityNotFoundException: Unable to f
ind explicit activity class {com.example.mislugares/com.example.m
islugares.VistaLugar}; have you declared this activity in your An
droidManifest.xml?
```

10. *Para resolver el error en ejecución, registra la nueva actividad en AndroidManifest.xml.*
11. *Ejecuta la aplicación y verifica que cuando seleccionas el icono buscar se arranca una actividad que muestra el primer lugar.*



Ejercicio: *Un cuadro de diálogo para indicar el* id *de lugar*

Tras realizar el ejercicio anterior, comprobarás que siempre se visualiza el lugar con posición 0. En este ejercicio vamos a introducir un cuadro de diálogo que permita introducir al usuario el *id* que desea visualizar.



Ha de quedar claro que esta es la forma más correcta de diseñar la interfaz de usuario. Más adelante reemplazaremos este cuadro de diálogo por un `RecyclerView`.

1. Abre la clase `MainActivity` del proyecto Mis Lugares.
2. Reemplaza el método por el `lanzarVistaLugar()` siguiente:


```
public void lanzarVistaLugar(View view){
   final EditText entrada = new EditText(this);
   entrada.setText("0");
   new AlertDialog.Builder(this)
      .setTitle("Selección de lugar")
      .setMessage("indica su id:")
      .setView(entrada)
      .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
         public void onClick(DialogInterface dialog, int whichButton) {
            int id = Integer.parseInt(entrada.getText().toString());
            usoLugar.mostrar(id);
         }})
      .setNegativeButton("Cancelar", null)
      .show();
}
```

```
fun lanzarVistaLugar(view: View? = null) {
   val entrada = EditText(this)
   entrada.setText("0")
   AlertDialog.Builder(this)
      .setTitle("Selección de lugar")
      .setMessage("indica su id:")
      .setView(entrada)
      .setPositiveButton("Ok")  { dialog, whichButton ->
         val id = parseInt(entrada.text.toString())
         usoLugar.mostrar(id);
      }
      .setNegativeButton("Cancelar", null)
      .show()
}
```

Nota sobre Java / Kotlin: *Es posible crear un objeto sin que este disponga de un identificador de objeto. Este tipo de objeto se conoce como objeto anónimo. El código mostrado a continuación a la derecha es equivalente al de la izquierda.*

```
Clase objeto = new Clase();          new Clase().metodo();
objeto.metodo();
```

```
objeto: Clase = Clase()              Clase().metodo()
objeto.metodo();
```

Un objeto anónimo no tiene identificador, por lo que solo puede usarse donde se crea. En el método anterior se ha creado un objeto anónimo de la clase `AlertDialog.Builder`. *Observa cómo no se llama a un método, si no a una cadena de métodos. Esto es posible porque los métodos de la clase* `AlertDialog.Builder` *retornan el objeto que estamos creando. Por lo tanto, cada método se aplica al objeto devuelto por el método anterior.*

En Android puedes usar la clase `AlertDialog` para crear un cuadro de diálogo configurable. Si te fijas en la captura anterior, el cuadro de diálogo está formado

por cuatro elementos, de arriba abajo: *título*, *mensaje*, *vista* y *botones*. Estos elementos pueden configurarse mediante los método `setTitle()`, `setMessage()`, `setView()`, `setPositiveButton()` y `setNegativeButton()`.

La vista que se utiliza en este diálogo es un `EditText`, inicializado con un texto. En caso de necesitar varias entradas, se puede crear una vista de tipo *layout*, que contendría estas entradas. Se han introducido dos botones, indicando el texto del botón y un escuchador de evento al que se llamará cuando se pulse el botón. Finalmente se llama al método `show()` para que se visualice el cuadro de diálogo.

3. Verifica que funciona correctamente. Pero cuidado, no se verifica que el *id* sea válido, por lo que ocurrirá un error si es incorrecto.



Práctica: *Ocultar elementos en VistaLugarActivity*

En ocasiones no se dispondrá de parte de la información de un lugar. En estos casos, puede resultar más conveniente, desde un punto de vista estético, no mostrar campos sin información en la actividad `VistaLugarActivity`. Por ejemplo, si el campo de teléfono es igual a 0, podríamos usar el siguiente código para que no se muestre:

```
if (lugar.getTelefono() == 0) {
    findViewById(R.id.telefono).setVisibility(View.GONE);
} else {
    findViewById(R.id.telefono).setVisibility(View.VISIBLE);
    TextView telefono = findViewById(R.id.telefono);
    telefono.setText(Integer.toString(lugar.getTelefono()));
}
```

```
if (lugar.telefono == 0) {
   telefono.setVisibility(View.GONE)
} else {
   telefono.setVisibility(View.VISIBLE)
   telefono.setText(Integer.toString(lugar.telefono))
}
```

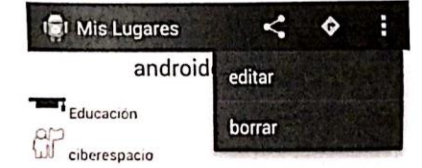
Para ocultarlo, en el *layout* `telefono`, ponemos el valor propiedad `visibility` al valor `GONE`. Este atributo se aplica a cualquier tipo de vista. Otros posibles valores para este atributo son `VISIBLE` e `INVISIBLE`. Tanto con `GONE` como con `INVISIBLE` la vista no se verá. Pero con `INVISIBLE` el espacio ocupado por la vista se mantiene, mientras que con `GONE` este espacio se elimina.

Trata de realizar un proceso similar a este para los campos `dirección`, `telefono`, `url` y `comentario`. Para verificar si un `String` es vacío puedes usar el método `isEmpty()`.



Ejercicio: *Añadir una barra de acciones a VistaLugarActivity*

En este ejercicio vamos a añadir a la barra de acciones de la actividad un menú similar al que se muestra a continuación:



1. En primer lugar, crea el fichero *res/menu/vista_lugar.xml*, que contendrá las acciones a mostrar. Para ello pulsa con el botón derecho sobre la carpeta *res/menu* y crea el fichero *vista_lugar*.
2. Reemplaza su contenido por el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:app="http://schemas.android.com/apk/res-auto">
   <item
        android:id="@+id/accion_compartir"
        android:title="compartir"
        android:icon="@android:drawable/ic_menu_share"
        android:orderInCategory="10"
        app:showAsAction="ifRoom"/>
   <item
        android:id="@+id/accion_llegar"
        android:title="cómo llegar"
        android:icon="@android:drawable/ic_menu_directions"
        android:orderInCategory="20"
        app:showAsAction="ifRoom"/>
   <item
        android:id="@+id/accion_editar"
        android:title="editar"
        android:icon="@android:drawable/ic_menu_edit"
        android:orderInCategory="30"
        app:showAsAction="ifRoom"/>
   <item
        android:id="@+id/accion_borrar"
        android:title="borrar"
        android:icon="@android:drawable/ic_menu_delete"
        android:orderInCategory="40"
        app:showAsAction="ifRoom"/>
</menu>
```

3. En la clase `VistaLugarActivity` añade los siguientes métodos:

```
@Override public boolean onCreateOptionsMenu(Menu menu) {
   getMenuInflater().inflate(R.menu.vista_lugar, menu);
```

```
   return true;
}
@Override public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
   case R.id.accion_compartir:
      return true;
   case R.id.accion_llegar:
      return true;
   case R.id.accion_editar:
      return true;
   case R.id.accion_borrar:
      usoLugar.borrar(pos);
      return true;
   default:
      return super.onOptionsItemSelected(item);
   }
}
```

```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
   menuInflater.inflate(R.menu.vista_lugar, menu)
   return true
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
   when (item.getItemId()) {
      R.id.accion_compartir -> return true
      R.id.accion_llegar -> return true
      R.id.accion_editar -> return true
      R.id.accion_borrar -> {
         usoLugar.borrar(pos)
         return true
      }
      else -> return super.onOptionsItemSelected(item)
   }
}
```

4. Añade a `CasosUsoLugar`:

```
public void borrar(int id) {
   lugares.borrar(id);
   actividad.finish();
}
```

```
fun borrar(id: Int) {
   lugares.borrar(id)
   actividad.finish()
}
```

5. Ejecuta la aplicación y borra un lugar. Verifica que, si tratas de visualizar el mismo *id*, ahora se muestra el siguiente lugar.



Práctica: Un cuadro de diálogo para confirmar el borrado

Un usuario podría pulsar por error el botón de borrar, por lo que sería muy conveniente pedir una confirmación antes de borrar.

1. En el método anterior, crea un cuadro de diálogo siguiendo el esquema planteado en el ejercicio anterior. Puede ser similar al siguiente.

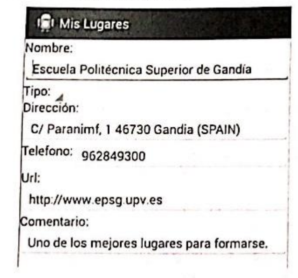
Borrado de lugar

¿Estás seguro que quieres eliminar este lugar?

Cancelar   Confirmar

3.6.2. Creando la actividad EdicionLugarActivity

En este apartado crearemos otra actividad en la aplicación Mis Lugares, `EdicionLugarActivity`. Esta actividad nos permitirá modificar la mayoría de los valores asignados a un lugar (se excluyen los valores que se modifican desde `VistaLugarActivity`: valoración, fecha y foto):





Práctica: *Creación de la actividad EdicionLugarActivity*

1. En el proyecto Mis Lugares verifica que existe el *layout edicion_lugar.xml*. En caso contrario, realiza la práctica "Creación de Mis Lugares y formulario de edición".
2. Crea la clase `EdicionLugarActivity` y haz que extienda de `AppCompatActivity`. Copia en esta clase los atributos y los métodos `onCreate()` y `actualizaVistas()` de la clase `VistaLugarActivity`.

3. **En Java**, añade los siguientes atributos a la clase:

```
private EditText nombre;
private Spinner tipo;
private EditText direccion;
private EditText telefono;
private EditText url;
private EditText comentario;
```

De esta forma, estos seis objetos serán accesibles desde cualquier método de la clase, en lugar de estar declarados solo en el método `onCreate()`. La creación e inicialización de los objetos `nombre`, `direccion`, `telefono`, `url` y `comentario` puede realizarse de forma similar al copiado. Pero ahora, no han de ser declarados en el método, al estar declarados como atributos. Por lo tanto, has de eliminar el `TextView` inicial de cada objeto.

En Kotlin este proceso se realiza automáticamente.

4. *Reemplaza* la vista a mostrar en `setContentView()` por `edicion_lugar`.
5. *El* paso de parámetros para obtener `pos` y `lugar` puede realizarse de la misma forma.
6. Si has realizado la práctica "Ocultar elementos en VistaLugarActivity", has de eliminar el código introducido. Por ejemplo, en el caso del campo del teléfono elimina el código tachado:

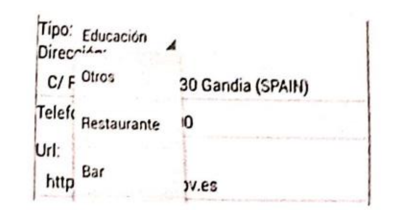
```
if (lugar.getTelefono() == 0) {
   findViewById(R.id.telefono).setVisibility(View.GONE);
} else {
   findViewById(R.id.telefono).setVisibility(View.VISIBLE);
   telefono = findViewById(R.id.telefono);
   telefono.setText(Integer.toString(lugar.getTelefono()));
}
```

7. Puedes eliminar el resto del código de este método, que hace referencia a `logo_tipo`, `tipo`, `fecha`, `hora` y `valoracion`.
8. Crea un caso de uso, con la función `editar(pos)` que abra la actividad que acabas de crear. Usa `mostrar(pos)` como referencia.
9. En la clase `VistaLugarActivity`, dentro del método `onOptionsItemSelected()`, añade el código necesario para que se llame a esta función.
10. Ejecuta el proyecto. Pero antes, piensa si falta alguna acción por realizar.



Ejercicio: *Inicializar el Spinner en EdicionLugarActivity*

Como has podido verificar en la ejecución anterior, el `Spinner` (lista desplegable) no muestra ningún valor. En este ejercicio trataremos de que funcione correctamente:



1. Añade el siguiente código al método `actualizaVistas()`:

```
tipo = findViewById(R.id.tipo);
ArrayAdapter<String> adaptador = new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item, TipoLugar.getNombres());
adaptador.setDropDownViewResource(android.R.layout.
        simple_spinner_dropdown_item);
tipo.setAdapter(adaptador);
tipo.setSelection(lugar.getTipo().ordinal());
```

```
val adaptador = ArrayAdapter<String>(this,
   android.R.layout.simple_spinner_item,
   lugar.tipoLugar.getNombres()
)
adaptador.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item)
tipo.adapter = adaptador
tipo.setSelection(lugar.tipoLugar.ordinal)
```

Para inicializar los valores que puede tomar un `Spinner`, necesitamos una clase especial conocida como `Adapter`. Esta clase se estudiará en la siguiente unidad. De momento, solo adelantamos que un `Adapter` va a crear una lista de vistas, inicializándolas con unos valores determinados. La clase `ArrayAdapter<String>` es un tipo de `Adapter` que permite inicializar sus valores a partir de un *array* de `String`. Su constructor necesita tres parámetros: un contexto (usamos la actividad actual), una vista para mostrar elemento (usamos un vista definida en el sistema) y un *array* de `String`. Para el último parámetro necesitamos un *array* con todos los valores, que puede tomar el enumerado `TipoLugar`. Para obtener este *array*, se define un nuevo método, que se muestra a continuación.

El siguiente método, `setDropDownViewResource()`, permite indicar una vista alternativa que se usará cuando se despliegue el `Spinner`. En la versión 4.x, esta vista es un poco más grande que la usada en el método anterior, para poder seleccionarla cómodamente con el dedo. Este código concluye asignando el adaptador al `Spinner` y poniendo un valor inicial según el tipo actual de lugar.

2. Añade el siguiente método a la clase `TipoLugar`:

```
public static String[] getNombres() {
   String[] resultado = new String[TipoLugar.values().length];
   for (TipoLugar tipo : TipoLugar.values()) {
      resultado[tipo.ordinal()] = tipo.texto;
   }
   return resultado;
}
```

```
…;
fun getNombres(): Array<String?> {
   val resultado = arrayOfNulls<String>(TipoLugar.values().size)
   for (tipo in TipoLugar.values()) {
      resultado[tipo.ordinal] = tipo.texto
   }
   return resultado
}
```

3. *Ejecuta la aplicación y verifica que la lista desplegable funciona correctamente.*



Práctica: *Añadir una barra de acciones a EdicionLugarActivity*

En esta práctica vamos a añadir a la actividad un menú en la barra de acciones similar al que se muestra a continuación:

Mis Lugares CANCELAR GUARDAR
Nombre:
Escuela Politécnica Superior de Gandia

1. *Crea un nuevo recurso de menú con las opciones que se indican.*
2. Asocia este menú a la actividad `EdicionLugarActivity` con el método `onCreateOptionsMenu()`.
3. Crea el método `onOptionsItemSelected()` de manera que cuando se seleccione la acción *Guardar* se ejecute el siguiente código:

```
lugar.setNombre(nombre.getText().toString());
lugar.setTipo(TipoLugar.values()[tipo.getSelectedItemPosition()]);
lugar.setDireccion(direccion.getText().toString());
lugar.setTelefono(Integer.parseInt(telefono.getText().toString()));
lugar.setUrl(url.getText().toString());
lugar.setComentario(comentario.getText().toString());
usoLugar.guardar(pos, lugar);
finish();
```

```
val nuevoLugar = Lugar(nombre.text.toString(), direccion.text.toString(),
         lugar.posicion, TipoLugar.values()[tipo.selectedItemPosition],
         lugar.foto, Integer.parseInt(telefono.text.toString()),
         url.text.toString(), comentario.text.toString(),
         lugar.fecha, lugar.valoracion )
usoLugar.guardar(pos, nuevoLugar)
finish()
```

4. Cuando se seleccione la acción cancelar, simplemente se saldrá de la actividad.
5. Añade a `CasosLugar` la siguiente función:

```
public void guardar(int id, Lugar nuevoLugar) {
   lugares.actualiza(id, nuevoLugar);
}
```



```
fun guardar(id: Int, nuevoLugar: Lugar) {
   lugares.actualiza(id, nuevoLugar)
}
```

6. Ejecuta la aplicación. Modifica algún lugar y pulsa *Guardar*. Al regresar a la actividad anterior, los valores permanecen sin variación. Sin embargo, si pulsas la tecla de volver y entras a visualizar el mismo lugar, los cambios sí que se actualizan. ¿Qué puede estar pasando?



Ejercicio: *Refrescar VistaLugarActivity tras entrar en EdicionLugarActivity*

Parece que al regresar a `VistaLugarActivity` desde `EdicionLugarActivity` no estamos indicando que vuelva a obtener los datos mostrados en las vistas. Para actualizar estos valores, puedes hacer los siguientes pasos:

1. Añade el código subrayado en `CasosUsoLugar`:

```
public void editar(int pos, int codidoSolicitud) {
   Intent i = new Intent(actividad, EdicionLugarActivity.class);
   i.putExtra("pos", pos);
   actividad.startActivityForResult(i, codidoSolicitud);
}
```

```
fun editar(pos: Int, codidoSolicitud: Int) {
   val i = Intent(actividad, EdicionLugarActivity::class.java)
   i.putExtra("pos", pos);
   actividad.startActivityForResult(i, codidoSolicitud)
}
```

2. Añade la siguiente constante a `VistaLugarActivity`:

```
final static int RESULTADO_EDITAR = 1;
```

```
val RESULTADO_EDITAR = 1
```

3. En el método `onOptionsItemSelected()` incluye el nuevo parámetro.
4. Añade el siguiente método a `VistaLugarActivity`:

```
@Override protected void onActivityResult(int requestCode, int resultCode,
                  Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   if (requestCode == RESULTADO_EDITAR) {
      actualizaVistas();
   }
}
```

```
override fun onActivityResult(requestCode: Int, resultCode: Int,
                  data: Intent?) {
   if (requestCode == RESULTADO_EDITAR) {
      actualizaVistas()
```

```
    }
}
```

Una vez regresamos de la actividad `EdicionLugarActivity`, lo que hacemos es actualizar los valores de las vistas y forzar al sistema a que repinte la vista con id `scrollView1`. Esta vista corresponde al `ScrollView` que contiene todo el *layout*.

3.7. Creación y uso de iconos

En el apartado anterior hemos creado una barra de acciones donde se mostraban algunos iconos. Se han utilizado iconos disponibles en el sistema Android, es decir, recursos ya almacenados en el sistema. Otra alternativa es crear tus propios iconos y almacenarlos en la carpeta `res/mipmap`*. En este apartado aprenderemos a hacerlo.*



Vídeo[tutorial]: *Creación de iconos en Android*

En Android se utilizan diferentes tipos de iconos según su utilidad. La siguiente tabla muestra los más importantes:

Tipo de iconos	Finalidad	Ejemplos
Lanzadores	Representa la aplicación en la pantalla principal y en la tienda de Apps.	
Barra de acciones	Opciones disponibles en la barra de acciones.	
Notificaciones	Pequeños iconos que aparecen en la barra de estado.	
Otros	También es muy frecuente el uso de iconos en cuadros de diálogo, *RecyclerView*, etc.	

Tabla 4: *Tipos de iconos en Android.*

El icono más importante de una aplicación es sin duda el icono lanzador o *launcher*. Se utiliza para identificar tu aplicación, tanto en el sistema Android como en la tienda de aplicaciones. El icono a utilizar ha de indicarse en un atributo de AndroidManifest:

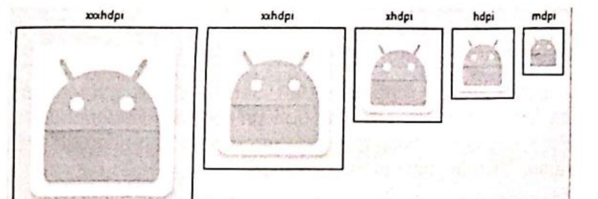
```
<application
    android:icon="@mipmap/ic_launcher"/>
```


Si vas a los recursos a ver como se define `ic_launcher`, verás que hay hasta 14 ficheros con este nombre. ¿Por qué tantos? La razón está en que a lo largo de las distintas versiones de Android han ido apareciendo hasta tres formas diferentes de crear este icono. Como queremos que el icono se vea bien en todas las versiones, tenemos que crearlo de varias formas. Veamos las tres formas de crear iconos en Android:

res
drawable
(3) ic_launcher_background.xml
ic_launcher_foreground.xml (v24)
mipmap
ic_launcher (6)
(1) ic_launcher.png (hdpi)
ic_launcher.png (mdpi)
ic_launcher.png (xhdpi)
ic_launcher.png (xxhdpi)
ic_launcher.png (xxxhdpi)
(3) ic_launcher.xml (anydpi-v26)
ic_launcher_round (6)
(2) ic_launcher_round.png (hdpi)
ic_launcher_round.png (mdpi)
ic_launcher_round.png (xhdpi)
ic_launcher_round.png (xxhdpi)
ic_launcher_round.png (xxxhdpi)
ic_launcher_round.xml (anydpi-v26)

Heredados - *legacy icons* (1): Disponibles desde la primera versión de Android. Los dispositivos con versiones anteriores a la 8.0 los utilizan. Si la versión mínima de API es anterior, tendremos que incluirlos en la aplicación.

Android ha sido concebido para ser utilizado en pantallas con una gran variedad de densidades gráficas. Este rango va desde 160 píxeles/pulgada (*mdpi*) hasta 640 píxeles/pulgada (*xxhdpi*). Sin embargo, el ancho del icono queremos que sea igual en todas las pantallas, en concreto 0,3 pulgadas. Por lo tanto, con una densidad de 160 ppp, necesitamos un icono de 0,3 · 160 = 48 píxeles de lado. Pero para 640 dpi, necesitamos uno de 0,3 · 640 = 192 píxeles. Accede en los recursos a *res/mipmap/ic_launcher* y observarás como se ha creado un icono para cada una de las densidades gráficas. Recuerda que la carpeta *mipmap* se utilizaba para recursos gráficos que no han de ser reescalados.



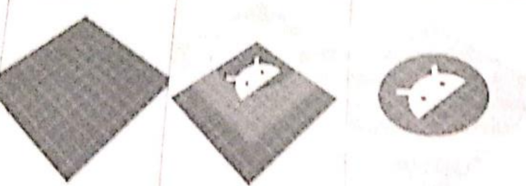
Circulares - round icons (2): Aparecen en la versión 7.1 (API 25) cuando muchas distribuciones de Android optaron por utilizar iconos circulares. Algunas, incluso, daban la posibilidad de escoger al usuario entre iconos normales y circulares:



Para que el icono de nuestra aplicación se vea uniforme con el resto de iconos, es interesante proporcionar los dos tipos de iconos. Para ello se ha añadido un nuevo atributo en AndroidManifest:

```
<application
    android:icon="@mipmap/ic_launcher"
    android:roundIcon="@mipmap/ic_launcher_round"... />
```

Adaptativos - adaptative Icons (3): Aparecen en la versión 8.0 (API 26) para dar mayor flexibilidad en la representación de los iconos. Para diseñar un icono habrá que proporcionar dos capas: fondo y primer plano; luego, se aplicará una máscara:



Nosotros proporcionaremos las dos capas, pero la máscara la pondrá el sistema según las preferencias del usuario:



Los iconos adaptativos se definen en un fichero xml de la siguiente forma:

```
<adaptive-icon xmlns:android="http://schemas.android.com/apk/res/android">
    <background android:drawable="@drawable/ic_launcher_background" />
    <foreground android:drawable="@drawable/ic_launcher_foreground" />
</adaptive-icon>
```

Verifica como ambos ficheros están dentro de la carpeta *drawable*. Son definidos de forma vectorial. Trabajar con un formato vectorial tiene importantes ventajas. Por ejemplo, los iconos pueden generarse con diferentes dimensiones o pueden girarse. Además, ocupan menos memoria.



Enlaces de interés:

Guía de estilo para iconos: *La siguiente página describe las guías de estilo para los iconos en Material Design:*

https://youtu.be/wt0Uzc9UHNw

https://material.io/design/iconography

Recursos de iconos: En las siguientes páginas puedes encontrar gran variedad de iconos:

https://material.io/tools/icons

https://android-material-icon-generator.bitdroid.de/



Ejercicio: *Creación de iconos personalizados*

Veamos un ejemplo práctico de cómo crear un icono. El diseñador gráfico de nuestra empresa nos acaba de pasar dos gráficos vectoriales para crear el icono de una aplicación. El primer plano es una estrella amarilla y el fondo es blanco con cuatro rallas azules.

1. Pulsa con el botón derecho sobre *res/drawable* y selecciona *New/Drawable Resource File*. Introduce en nombre *estrella.xml*. Reemplaza el contenido por:

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="60dp"
    android:height="100dp"
    android:viewportWidth="6.0"
    android:viewportHeight="10.0">
    <path android:fillColor="#D9A802"
        android:pathData=
        "M0,2L2,2L3,0L4,2L6,2L4.5,4L6,6L4,6L3,10L2,6L0,6L1.5,4Z"/>
</vector>
```

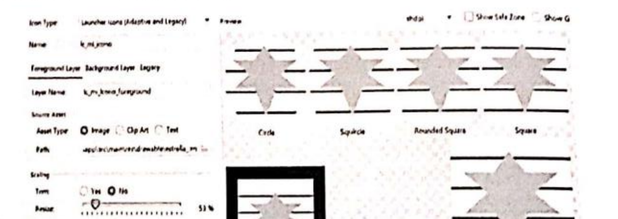
NOTA: El objetivo de esta unidad no es describir los gráficos vectoriales, pero, si tienes curiosidad, te damos algunas claves para que entiendas este fichero. Primero se indica el ancho y alto que tendrá el gráfico y luego, el ancho y alto con el que lo vamos a dibujarlo. La etiqueta path *permite realizar un trazado rellenándolo del color indicado. Nos movemos a la coordenada 0,2 (*M0,2*); trazamos una línea hasta 2,2 (*L2,2*); otra línea hasta 3,0 (*L3,0*). Se sigue trazando líneas hasta llegar a Z, que significa trazar una línea hasta el primer punto (0,2) y rellena la figura.*

2. Repite el proceso para el fichero *fondo.xml*.

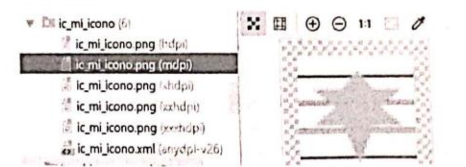
```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="108dp"
    android:height="108dp"
    android:viewportWidth="5.0"
    android:viewportHeight="5.0">
    <path android:fillColor="#FFFFFF"
        android:pathData="M0,0h5v5h-5z"/>
    <path android:strokeColor="#394077"
        android:strokeWidth="0.1"
        android:pathData="M0,1L5,1 M0,2L5,2 M0,3L5,3 M0,4L5,4"/>
</vector>
```

NOTA: Este fichero dibuja dos path*. Un rectángulo blanco que ocupa toda el área de dibujo y cuatro líneas horizontales de color azul.*

3. Selecciona *File/New/Image Asset*. Selecciona en *Icon Type: Launcher Icons (Adaptive and Legacy)*; en *Name: ic_mi_icono*; en *Asset Type: Image*; en *Path:* pulsa el icono de la carpeta, en la parte superior pulsa en el icono de Android, para buscar un fichero en tu proyecto. Selecciona *app\ src\ main\ res\ drawable\ estrella.xml*. Desplaza la barra *Resize* hasta conseguir un tamaño adecuado.
4. Selecciona la lengüeta Background Layer. Selecciona en Asset Type: Image; en Path: pulsa el icono de la carpeta y repite el proceso anterior para seleccionar *fondo.xml*. Desplaza la barra *Resize* hasta conseguir un tamaño adecuado. El resultado ha de ser similar al siguiente:

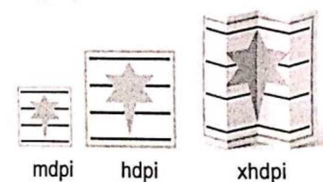


5. Pulsa en *Next* y se mostrará una previsualización de todos los ficheros generados. Pulsa en *Finish* para acabar.
6. Verifica como se han creado en res/mipmap/ic_mi_icono cinco gráficos png a partir del gráfico vectorial. Selecciona el de densidad mdpi. El resultado no ha sido 100 % satisfactorio.



Las líneas horizontales tendrían que ser del mismo grosor. Para ver si el usuario llegará a apreciarlo, reduce el zoom hasta que el icono tenga un tamaño similar al que tendrá en un móvil. Si sigues apreciando que las rallas son diferentes, es buena idea retocar este fichero. Abre el png con un editor gráfico y redibuja las líneas horizontales para que tenga el mismo grosor.

Aunque este no es el caso, algunos iconos tienen muchos detalles. Cuando se representan para mdpi con tan solo 48x48 píxeles, puede ocurrir que se vea de forma incorrecta. En baja densidad se recomienda cambiar a un diseño más simple. Mira el siguiente ejemplo:



7. Verifica como se han creado en *res/mipmap/ic_mi_icono_round* cinco gráficos png con forma circular.
8. Verifica como se ha creado el icono adaptativo en *res/mipmap/ic_mi_icono.xml*. Comprueba también como se han creado las dos capas que corresponden a versiones redimensionadas de los ficheros *estrella.xml* y *fondo.xml*.
9. Para aplicar este icono como el lanzador de tu aplicación, modifica el fichero AndroidManifest:

```
<application
    android:icon="@mipmap/ic_mi_icono"
    android:roundIcon="@mipmap/ic_mi_icono"… />
```

10. Ejecuta el proyecto y comprueba que el icono asociado es correcto.



Preguntas de repaso: *Iconos*

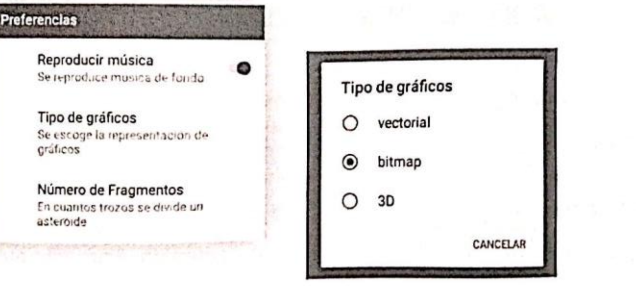
3.8. Añadiendo preferencias de usuario

Android nos facilita la configuración de nuestros programas, al permitir añadir una lista de preferencias que el usuario podrá modificar. Por ejemplo, el usuario podrá indicar con qué frecuencia la aplicación ha de sincronizarse con el servidor o si queremos que se lancen notificaciones. Las preferencias también pueden utilizarse para que tu aplicación almacene información de forma permanente. En el capítulo 9 se estudiará cómo realizar esta función.



Vídeo[tutorial]: *Añadir preferencias en Android*

NOTA: A continuación, se proponen una serie de ejercicios para añadir preferencias en Mis Lugares. Para hacerlo en Asteroides puedes cambiar los textos que aparecen, para que correspondan con la siguiente captura:



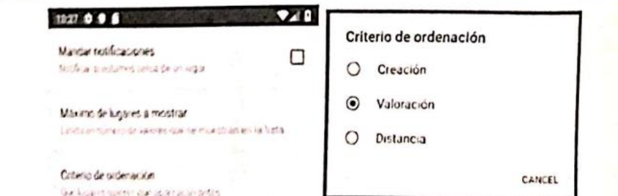


Ejercicio: *Añadiendo preferencias en la aplicación*

1. Abre el proyecto Mis Lugares (o Asteroides).
2. Pulsa con el botón derecho sobre la carpeta *res* y selecciona la opción *New > Android resource file*.
3. Completa los campos *File name: preferencias* y *Resource type*: XML. Se creará el fichero *res/xml/preferencias.xml*.
4. Edita este fichero. Selecciona la lengüeta *Code* e introduce el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.preference.PreferenceScreen
    xmlns:android="http://schemas.android.com/apk/res/android">
    <SwitchPreferenceCompat
        android:key="notificaciones"
        android:title="Mandar notificaciones"
        android:summary="Notificar si estamos cerca de un lugar"/>
    <EditTextPreference
        android:key="maximo"
        android:title="Máximo de lugares a mostrar"
        android:summary="Limita en número de valores que se muestran"
        android:inputType="number"
        android:defaultValue="12"/>
    <ListPreference
        android:key="orden"
        android:title="Criterio de ordenación"
        android:summary="Qué lugares quieres que aparezcan antes"
        android:entries="@array/tiposOrden"
        android:entryValues="@array/tiposOrdenValores"
        android:defaultValue="0"/>
</androidx.preference.PreferenceScreen>
```

El significado de cada etiqueta y atributo se descubre fácilmente si observas el resultado obtenido que se muestra a continuación. El atributo `inputType` permite configurar el tipo de teclado que se mostrará para introducir el valor. Coinciden con el atributo de `EditText`. Para ver los posibles valores consultar developer.android.com/reference/android/widget/TextView.html#attr_android:inputType.



5. Para almacenar los valores del desplegable, has de crear el fichero */res/values/arrays.xml* con el siguiente contenido. Para ello pulsa con el botón derecho sobre la carpeta *res* y selecciona la opción *New > Android resource file*. Completa los campos *File name: arrays* y *Resource type*: Values.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string-array name="tiposOrden">
        <item>Creación</item>
        <item>Valoración</item>
        <item>Distancia</item>
    </string-array>
    <string-array name="tiposOrdenValores">
        <item>0</item>
        <item>1</item>
        <item>2</item>
    </string-array>
</resources>
```

6. Añade al fichero *Gradle Scripts/Bulid.gradle(Module:app)* la dependencia:

```
dependencies {
   …
   implementation 'androidx.preference:preference:1.1.1'
}
```

7. Crea una nueva clase *PreferenciasFragment* con el siguiente código:

```
public class PreferenciasFragment extends PreferenceFragmentCompat {
   @Override
   public void onCreatePreferences(Bundle savedInstanceState,
                                   String rootKey) {
      setPreferencesFromResource(R.xml.preferencias, rootKey);
   }
}
```

```
class PreferenciasFragment : PreferenceFragmentCompat() {
  override fun onCreatePreferences(savedInstanceState: Bundle?,
                                   rootKey: String?) {
    setPreferencesFromResource(R.xml.preferencias, rootKey)
  }
}
```



Nota sobre Java/Kotlin: *Pulsa **Alt-Intro** para que automáticamente se añadan los* imports *con los paquetes que faltan.*

La clase `PreferenceFragmentCompat` permite crear un fragment que contiene una ventana con las opciones de preferencias definidas en un recurso XML. Un fragment es un elemento que puede ser incrustado dentro de una actividad. El uso de fragment se estudia con más detalle en el anexo A.

8. Ahora vamos a crear una actividad que simplemente muestre el fragment anterior. Crea la clase `PreferenciasActivity` con el siguiente código:

```java
public class PreferenciasActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getSupportFragmentManager().beginTransaction()
                .replace(android.R.id.content, new PreferenciasFragment())
                .commit();
    }
}
```

```kotlin
class PreferenciasActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      supportFragmentManager.beginTransaction()
         .replace(android.R.id.content, PreferenciasFragment())
         .commit()
   }
}
```

Desde una actividad podemos visualizar un fragment en tiempo de ejecución. Para ello utilizamos el manejador de fragments de la actividad (`getSupportFragmentManager()`) y comenzamos una transacción (`beginTransaction()`). Una transacción es una operación de insertado, borrado o reemplazo de fragments. En el ejemplo vamos a reemplazar el contenido de la actividad por un nuevo fragment de la clase `PreferenciasFragment`. Finalmente se llama a `commit()` para que se ejecute la transacción.

9. No hay que olvidar registrar toda nueva actividad en `AndroidManifest.xml`.
10. Añade a *MainActivity.java* el método `lanzarPreferencias()`. Este método ha de tener el mismo código que `lanzarAcercaDe()`, pero lanzando la actividad `PreferenciasActivity`. En el *layout activity_main.xml* añade al botón con el texto «Configurar» en el atributo `onClick` el valor `lanzarPreferencias`.
11. Para activar la configuración desde la opción de menú, añade el siguiente código en el fichero *MainActivity.java* en el método `onOptionsItemSelected()`:

```java
if (id == R.id.action_settings) {
   lanzarPreferencias(null);
   return true;
}
```

```kotlin
R.id.action_settings -> {
   lanzarPreferencias()
   true
}
```

Si has hecho la práctica "Casos de Uso para arrancar actividades" utiliza mejor el código `usoActividades.lanzarPreferencias()` para lanzar la actividad.

12. Arranca la aplicación y verifica que puedes lanzar las preferencias mediante las dos alternativas.

NOTA: Si introduces un valor en el campo donde se ha indicado inputType="number", comprobarás que no funciona correctamente. Se trata de un bug de Google, todavía sin resolver. De momento nos recomiendan que usemos la siguiente librería https://github.com/Gericop/Android-Support-Preference-V7-Fix. Sigue los dos sencillos pasos indicados en README.md.

3.8.1. Organizando preferencias<opcional>

Cuando el número de preferencias es grande, resulta interesante organizarlas de forma adecuada. Una posibilidad consiste en dividirlas en varias pantallas, de forma que cuando se seleccione una opción en la primera pantalla, se abra una nueva pantalla de preferencias. Para organizar las preferencias de esta forma, usa el siguiente esquema:

```xml
<PreferenceScreen>
  <CheckBoxPreference …/>
  <EditTextPreference …/>
  …
  <PreferenceScreen android:title="Modo multijugador">
    <CheckBoxPreference …/>
    <EditTextPreference …/>
    …
  </PreferenceScreen>
</PreferenceScreen>
```

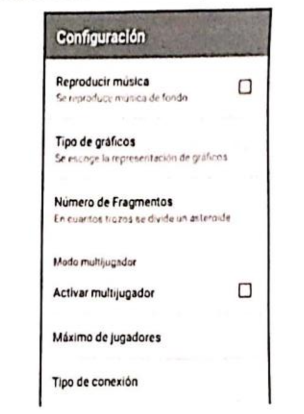
Práctica: *Organizando preferencias (I)*

1. Crea una nueva lista de preferencias `<PreferenceScreen>` dentro de la lista de preferencias del fichero *res/xml/preferencias.xml*.
2. Asígnale al parámetro `android:title` el valor "Modo multijugador".
3. Crea tres elementos dentro de esta lista: *Activar multijugador*, *Máximo de jugadores* y *Tipo de conexión*. Para este último, han de poder escogerse los valores: *Bluetooth*, *Wi-Fi* e *Internet*.

Otra alternativa para organizar las preferencias consiste en agruparlas por categorías. Con esta opción se visualizarán en la misma pantalla, pero separadas por grupos. Has de seguir el siguiente esquema:

```xml
<PreferenceScreen>
  <CheckBoxPreference …/>
  <EditTextPreference …/>
  …
  <PreferenceCategory android:title="Modo multijugador">
    <CheckBoxPreference …/>
    <EditTextPreference …/>
  . . …
  </PreferenceCategory>
</PreferenceScreen>
```

A continuación, se representa la forma en que se muestran las categorías:





Práctica: *Organizando preferencias (II)*

Modifica la práctica anterior para que, en lugar de mostrar las propiedades en dos pantallas, las muestre en una sola, tal y como se muestra en la imagen anterior.

3.8.2. Cómo se almacenan las preferencias de usuario<opcional>

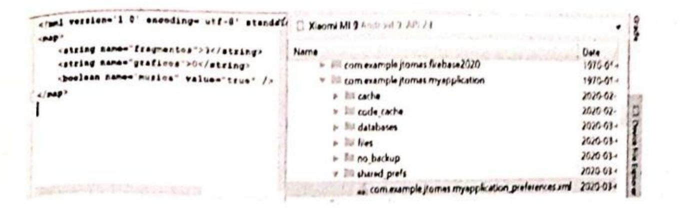
Si un usuario modifica el valor de una preferencia, este quedará almacenado de forma permanente en el dispositivo. Para conseguir esta persistencia, Android almacena las preferencias seleccionadas en un fichero XML dentro de la carpeta *data/data/*`nombre.del.paquete`*/files/shared_prefs*, donde `nombre.del.paquete` ha de ser reemplazado por el paquete de la aplicación. El nombre del fichero para almacenar las preferencias de usuario ha de ser siempre `nombre.del.paquete`*_preferences.xml*. Esto significa que solo puede haber unas preferencias de usuario por aplicación. Como se estudiará en el capítulo 9, puede haber otros ficheros de preferencias; pero, a diferencia de las preferencias de usuario, no pueden ser editadas directamente por el usuario, si no que hay que acceder a ellas por código.



Ejercicio: *Dónde se almacenan las preferencias de usuario*

Veamos dónde se han almacenado las preferencias que acabamos de crear:

1. Para navegar por el sistema de ficheros del dispositivo, desde Android Studio, selecciona la lengüeta *Device File Explorer* en la esquina inferior derecha.
2. Busca el siguiente fichero: */data/data/<nombre del paquete>/shared_prefs/ <nombre del paquete>_preferences.xml*



3. Haz doble clic sobre el fichero para visualizar su contenido:

```
<map>
    <boolean name="notificaciones" value="true" />
    <string name="maximo">12</string>
    <string name="orden">0</string>
</map>
```

3.8.3. Accediendo a los valores de las preferencias

Por supuesto, será necesario acceder a los valores de las preferencias para alterar el funcionamiento de nuestra aplicación. El siguiente ejemplo nos muestra cómo realizarlo:

```
public void mostrarPreferencias(){
   SharedPreferences pref =
                PreferenceManager.getDefaultSharedPreferences(this);
   String s = "música: " + pref.getBoolean("musica",true)
         +", gráficos: " + pref.getString("graficos","?");
   Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
}
```

El código comienza creando el objeto `pref` de la clase `SharedPreferences` y le asigna las preferencias definidas para la aplicación. A continuación crea el `String` `s` y le asigna los valores de dos de las preferencias. Se utilizan los métodos `pref.getBoolean()` y `pref.getString()`, que disponen de dos parámetros: el valor de `key` que queremos buscar (`"musica"` y `"graficos"`) y el valor asignado por defecto en caso de no encontrar esta `key`.

Finalmente se visualiza el resultado utilizando la clase `Toast`. Los tres parámetros indicados son el contexto (nuestra actividad), el `String` a mostrar y el tiempo que se estará mostrando esta información.



Ejercicio: *Accediendo a los valores de las preferencias*

1. Copia la función anterior en la actividad principal. Añade el parámetro que se muestra a continuación: `mostrarPreferencias(View view)`.
2. Asigna el atributo `onClick` del botón *Salir* o *Jugar* al método anterior, según estés en Mis Lugares o Asteroides.
3. Visualiza también el resto de las preferencias que hayas introducido.

3.8.4. Verificar valores correctos<opcional>

En muchas ocasiones vas a querer limitar los valores que un usuario puede introducir en las preferencias. Por ejemplo, podría ser interesante que el valor introducido por el usuario en la preferencia número de fragmentos solo pudiera tomar valores entre 0 y 9. Para conseguir esto podemos utilizar el escuchador de evento `onPreferenceChangeListener` que podremos asignar a una preferencia. Veamos cómo actuar en el siguiente ejercicio:



Ejercicio: *Verificar valores correctos de una preferencia*

1. Copia este código al final del método `onCreate()`en `PreferenciasFragment`:

```
final EditTextPreference fragmentos = (EditTextPreference)
                                        findPreference("maximo");
fragmentos.setOnPreferenceChangeListener(
  new Preference.OnPreferenceChangeListener() {
    @Override
    public boolean onPreferenceChange(Preference preference, Object
                                                        newValue) {
       int valor;
       try {
          valor = Integer.parseInt((String)newValue);
       } catch(Exception e) {
          Toast.makeText(getActivity(), "Ha de ser un número",
                                        Toast.LENGTH_SHORT).show();
          return false;
       }
       if (valor>=0 && valor<=99) {
          fragmentos.setSummary(
              "Limita en número de valores que se muestran ("+valor+")");
          return true;
       } else {
          Toast.makeText(getActivity(), "Valor Máximo 99",
                                        Toast.LENGTH_SHORT).show();
          return false;
       }
    }
  });
```

NOTA: Si trabajas con Asteroides reemplaza "maximo" por "fragmentos" y cambia los textos necesarios para que correspondan con esta propiedad.

El código comienza obteniendo una referencia de la preferencia, para asignarle un escuchador que será llamado cuando cambie su valor. El escuchador comienza convirtiendo el valor introducido a entero. Si se produce un error es porque el usuario no ha introducido un valor adecuado. En este caso, mostramos un mensaje y devolvemos `false` para que el valor de la preferencia no sea modificado. Si no hay error, tras verificar el rango de valores aceptables, modificamos la explicación de la preferencia para que aparezca el nuevo valor entre paréntesis y devolvemos `true` para aceptar este valor. Si no está en el rango, mostramos un mensaje indicando el problema y devolvemos `false`.

2. Ejecuta el proyecto y verifica que funciona correctamente.

Práctica: *Mostrar el valor de una preferencia*

En el ejercicio anterior cuando se modifica el número de fragmentos se muestra entre paréntesis el nuevo valor introducido. El funcionamiento no es del todo correcto, cuando entramos por primera vez o cuando se cambia el teléfono de vertical a horizontal este valor no se muestra. Añade el código necesario para que este valor aparezca siempre.



Preguntas de repaso: *Preferencias de usuario*

3.9. Añadiendo una lista de puntaciones en Asteroides

Muchos videojuegos permiten recordar las puntuaciones de partidas anteriores; de esta forma, un jugador puede tratar de superar su propio récord o mejorar el de otros jugadores.

En el capítulo 9 estudiaremos varios métodos para que esta información se almacene permanentemente en el sistema. En el capítulo 10 estudiaremos cómo podemos compartir esta información utilizando Internet. En este capítulo nos centraremos en representar una lista de puntuaciones de forma atractiva utilizando la vista `RecyclerView`.

Vamos a intentar que el mecanismo de acceso a esta lista de puntuaciones sea lo más independiente posible del método final escogido para almacenar la información. Con este propósito, vamos a definir la interfaz `AlmacenPuntuaciones`.



Ejercicio: *La interfaz AlmacenPuntuaciones*

1. Abre la aplicación Asteroides.
2. Pulsa con el botón derecho sobre la carpeta de código (*org.example.asteroides*) y selecciona *New* > *Java Class*.
3. En el campo *Name* introduce `AlmacenPuntuaciones`, en al campo *Kind* introduce `Interface` y pulsa *OK*.
4. Introduce el código que se muestra a continuación:

```
public interface AlmacenPuntuaciones {
  public void guardarPuntuacion(int puntos,String nombre,long fecha);
```

```
public List<String> listaPuntuaciones(int cantidad);
}
```



Nota sobre Java/Kotlin: *Una interfaz es una clase abstracta pura, es decir, una clase donde se indican los métodos, pero no se implementa ninguno (en este caso se dice que los métodos son abstractos). Permite al programador establecer una estructura que ha de seguir toda clase que implemente esta interfaz. En la interfaz solo se indican los nombres de los métodos, sus parámetros y tipos que retornan, pero no el código de cada método. Una interfaz también puede contener constantes, es decir, campos de tipo static y final.*

Las diferentes clases que definamos para almacenar puntuaciones van a implementar esta interfaz. Como ves, tiene dos métodos. El primero es para guardar la puntuación de una partida, con los parámetros: puntuación obtenida, nombre del jugador y fecha de la partida. El segundo es para obtener una lista de puntuaciones previamente almacenadas. El parámetro `cantidad` indica el número máximo de puntuaciones que ha de devolver.

5. Veamos a continuación una clase que utiliza esta interfaz. Para ello crea en el proyecto la clase `AlmacenPuntuacionesList`.

6. Introduce el siguiente código:

```
public class AlmacenPuntuacionesList implements AlmacenPuntuaciones{

   private List<String> puntuaciones;

   public AlmacenPuntuacionesList() {
      puntuaciones = new ArrayList<String>();
      puntuaciones.add("123000 Pepito Domingez");
      puntuaciones.add("111000 Pedro Martinez");
      puntuaciones.add("011000 Paco Pérez");
   }

   @Override public void guardarPuntuacion(int puntos,
                                 String nombre, long fecha) {
      puntuaciones.add(0, puntos + " " + nombre);
   }

   @Override public List<String> listaPuntuaciones(int cantidad) {
      return puntuaciones;
   }
}
```

Esta clase almacena la lista de puntuaciones. Tiene el inconveniente de que, al tratarse de una variable local, cada vez que se cierre la aplicación se perderán las puntuaciones. El constructor inicializa *la lista* e introduce tres valores. La idea es que, aunque todavía no esté programado el juego y no podamos jugar, tengamos ya algunas puntuaciones para poder representar una lista. El método `guardarPuntuacion()` se limita a insertar en la primera posición de *la lista* un `String` con los puntos y el nombre. La fecha no se almacena. El método

`listaPuntuaciones()` devuelve la lista de `String` entero, sin tener en cuenta el parámetro `cantidad`, que debería limitar el número de *strings* devueltos.

7. En la actividad `MainActivity` tendrás que declarar una variable para almacenar las puntuaciones:

```
public static AlmacenPuntuaciones almacen= new AlmacenPuntuacionesList();
```



Nota sobre Java: *El modificador* `static` *permite compartir el valor de una variable entre todos los objetos de la clase. Es decir, aunque se creen varios objetos, solo existirá una única variable* `almacen` *compartida por todos los objetos. El modificador* `public` *permite acceder a la variable desde fuera de la clase. Por lo tanto, no será necesario crear métodos getters y setters. Para acceder a esta variable, no tendremos más que escribir el nombre de la clase seguida de un punto y el nombre de la variable. Es decir,* `MainActivity.almacen`.

8. Para que los jugadores puedan ver las últimas puntuaciones obtenidas, modifica el cuarto botón del *layout activity_main.xml* para que en lugar del texto "Salir" se visualice "Puntuaciones". Para ello modifica los ficheros *res/values/strings*. También sería interesante que cambiaras el fichero *res/values-en/strings*.

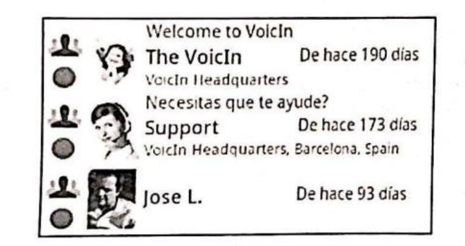
9. Modifica el escuchador asociado al cuarto botón para que llame al método:

```
public void lanzarPuntuaciones(View view) {
   Intent i = new Intent(this, Puntuaciones.class);
   startActivity(i);
}
```

10. De momento no te permitirá ejecutar la aplicación. Hasta que en el siguiente apartado no creemos la actividad `Puntuaciones`, no será posible.

3.10. Creación de listas con RecyclerView

La vista `RecyclerView` visualiza una lista o cuadrícula deslizable de varios elementos, donde cada elemento puede definirse mediante un *layout*. Su utilización es algo compleja, pero muy potente. Un ejemplo lo podemos ver en la siguiente figura:



Dentro del API de Android encontramos las vistas `ListView` y `GridView` que nos ofrecen una alternativa a `RecyclerView`. Esta última no ha sido añadida a ningún

API: se añade a una librería de compatibilidad. A pesar de que resulta algo más compleja de manejar, recomendamos el uso de `RecyclerView`, en lugar de `ListView` o `GridView`, al ser más eficiente y flexible. Aunque su uso se describe con detalle en *El Gran Libro de Android Avanzado*, hacemos en este punto una introducción de sus funcionalidades básicas. Las principales ventajas que ofrece `RecyclerView` frente a `ListView` o `GridView` son:

- Reciclado de vistas (`RecyclerView.ViewHolder`)
- Distribución de vistas configurable (`LayoutManager`)
- Animaciones automáticas (`ItemAnimator`)
- Separadores de elementos (`ItemDecoration`)
- Trabaja conjuntamente con otros widgets introducidos en Material Design (`CoordinationLayout`)



Vídeo[tutorial]: *Creación de listas con RecyclerView*

Crear una lista (o cuadrícula) de elementos con un un `RecyclerView` conlleva los siguientes pasos:

1. Diseñar un Layout que contiene el `RecyclerView`.
2. Implementar la actividad que visualice el `RecyclerView`.
3. Diseñar un *layout* individual que se repetirá en la lista.
4. Personalizar cada una de los *layouts* individuales según nuestros datos utilizando un adaptador.
5. Definir como queremos que se posicionen los elementos en las vistas. Por ejemplo, en forma de lista o de cuadricula.

Los tres primeros pasos anteriores son similares al uso de cualquier otro tipo de vista. Los dos últimos sí que requieren una explicación más extensa:

Personalizar los datos a mostrar

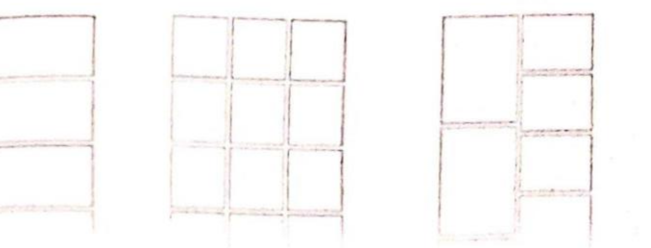
Para personalizar los elementos a mostrar en un `RecyclerView` hemos de usar un adaptador. La creación de adaptadores puede ser delicada, en algunos casos podemos tener problemas de eficiencia. Para evitar estos problemas, Google ha cambiado la forma de trabajar con `RecyclerView`. Ya no se puede utilizar la interfaz `Adapter`, si no que se ha de utilizar la clase `RecyclerView.Adapter`.



Vídeo[tutorial]: *El patrón ViewHolder y su uso en un RecyclerView*

Distribuir los elementos

A diferencia `ListView` o `GridView`, que muestran los elementos usando una determinada configuración, `RecyclerView` puede configurar esta distribución por medio de la clase `LayoutManager`. El sistema nos proporciona tres descendientes de `LayoutManager`, que son mostrados en la siguiente figura. También podemos crear nuestro descendiente de `LayoutManager`.



`LinearLayoutManager`   `GridLayoutManager`   `StaggeredGridLayoutManager`

En los siguientes ejercicios usaremos un `RecyclerView` en Asteroides para mostrar un listado de las puntuaciones.


Ejercicio: *Un RecyclerView en Asteroides*

1. Añade al fichero *Gradle Scripts/Bulid.gradle(Module:app)* la dependencia:

```
dependencies {
   …
   implementation 'androidx.recyclerview:recyclerview:1.1.0'
}
```

NOTA: Si lo deseas, puedes saltarte este paso. Más adelante, cuando aparezca RecyclerView en el código Java, la clase aparecerá marcada en rojo, al no encontrar su declaración. Al pulsar sobre la clase, aparecerá una bombilla roja donde podrás elegir la opción Add dependency on androidx.recyclerview:recyclerview. Él mismo añadirá la dependencia, con la ventaja de seleccionar la última versión disponible.

La clase `RecyclerView` no ha sido añadida a ninguna API de Android, sino que se encuentra en una librería externa. Esto tiene la ventaja de que, aunque esta clase aparece en la versión 5 de Android, puede ser usada en versiones anteriores. Crea un nuevo *layout* que se llame *puntuaciones.xml* con el siguiente código:

```
<androidx.recyclerview.widget.RecyclerView
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/recyclerView"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:layout_marginLeft="10dp"
   android:layout_marginRight="10dp"/>
```

2. Crea la clase `Puntuaciones` con el siguiente código:

```
public class Puntuaciones extends AppCompatActivity {
    private RecyclerView recyclerView;
    private RecyclerView.LayoutManager layoutManager;
    private MiAdaptador adaptador;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.puntuaciones);
        recyclerView = findViewById(R.id.recyclerView);
        adaptador = new MiAdaptador(this,
                        MainActivity.almacen.listaPuntuaciones(10));
        recyclerView.setAdapter(adaptador);
        layoutManager = new LinearLayoutManager(this);
        recyclerView.setLayoutManager(layoutManager);
    }
```

La actividad `Puntuaciones` se limita a visualizar el `RecyclerView`. Tras llamar al `super` y asignarle el layout, se buscar el `RecyclerView` por medio de su identificador. Creamos un adaptador y se lo asignamos al `RecyclerView`. La clase `MiAdaptador` será definida a continuación. Además, creamos un nuevo `LayoutManager` de tipo `LinearLayoutManager` y lo asignamos al `RecyclerView`.

3. Ahora hemos de definir el layout que representará cada uno de los elementos de la lista. Crea el fichero *res/layout/elemento_lista.xml* con el siguiente código:

```
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/relativeLayout"
    android:layout_width="match_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:paddingTop="4dp"
    android:paddingBottom="4dp">
    <ImageView
        android:id="@+id/icono"
        android:layout_width="?android:attr/listPreferredItemHeight"
        android:layout_height="match_parent"
        android:src="@drawable/asteroide2"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
    <TextView
        android:id="@+id/titulo"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:maxLines="1"
        android:textAppearance="?android:attr/textAppearanceLarge"
        app:layout_constraintStart_toEndOf="@+id/icono"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```



```
    <TextView
        android:id="@+id/subtitulo"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:gravity="center"
        android:text="Otro Texto"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toEndOf="@+id/icono"
        app:layout_constraintTop_toBottomOf="@+id/titulo" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Este *layout* representa una imagen a la izquierda con dos textos a la derecha, uno de mayor tamaño en la parte superior. Para combinar estos elementos se ha escogido un `ConstraintLayout`, donde la altura se establece a partir de un parámetro de configuración del sistema `?android:attr/listPreferredItemHeight`. El primer elemento que contiene es un `ImageView` alineado a la izquierda. Su altura es la misma que el contenedor (`match_parent`), mientras que la anchura se establece con el mismo parámetro que la altura del contenedor. Por lo tanto, la imagen será cuadrada.

A la derecha se muestran dos textos. En el texto de mayor tamaño se visualizará la puntuación y en el de menor tamaño un texto fijo.

4. Copia los ficheros *asteroide1.png* y *asteroide3.png* en la carpeta *res/drawable*. Puedes encontrar los gráficos en el siguiente link:

   http://www.androidcurso.com/images/dcomg/ficheros/Graficos.zip

5. El siguiente paso será crear la clase *MiAdaptador.java*, que se encargará de rellenar el `RecyclerView`:

```
public class MiAdaptador extends
                         RecyclerView.Adapter<MiAdaptador.ViewHolder> {
  private LayoutInflater inflador;
  private List<String> lista;

  public MiAdaptador(Context context, List<String> lista) {
    this.lista = lista;
    inflador =(LayoutInflater)
              context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
  }

  @Override
  public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View v = inflador.inflate(R.layout.elemento_lista, parent, false);
    return new ViewHolder(v);
  }

  @Override
  public void onBindViewHolder(ViewHolder holder, int i) {
    holder.titulo.setText(lista.get(i));
    switch (Math.round((float)Math.random()*3)){
      case 0:
        holder.icon.setImageResource(R.drawable.asteroide1);
        break;
```

```
        case 1:
            holder.icon.setImageResource(R.drawable.asteroide2);
            break;
        default:
            holder.icon.setImageResource(R.drawable.asteroide3);
            break;
    }
}

@Override
public int getItemCount() {
    return lista.size();
}

public class ViewHolder extends RecyclerView.ViewHolder {
    public TextView titulo, subtitutlo;
    public ImageView icon;

    ViewHolder(View itemView) {
        super(itemView);
        titulo = itemView.findViewById(R.id.titulo);
        subtitutlo = itemView.findViewById(R.id.subtitulo);
        icon = itemView.findViewById(R.id.icono);
    }
}
}
```

Un adaptador es un mecanismo estándar en Android que nos permite crear una serie de vistas que han de ser mostradas dentro de un contenedor. Con `RecyclerView` has de heredar de la clase `RecyclerView.Adapter` para crear el adaptador.

En el constructor se inicializa el conjunto de datos a mostrar (en el ejemplo un vector de puntuaciones) y el inflador nos va a permitir crear una vista a partir de su XML.

Luego se crea la clase `ViewHolder`, que contendrá las vistas que queremos modificar de un elemento (en concreto: dos `TextView` con el título y el subtítulo y un `ImageView` con la imagen). Esta clase es utilizada para evitar tener que crear las vistas de cada elemento desde cero. Lo va a hacer es utilizar un `ViewHolder` que contendrá las tres vistas ya creadas, pero sin personalizar. De forma que, gastará el mismo `ViewHolder` para todos los elementos y simplemente lo personalizaremos según la posición. Es decir, reciclamos el `ViewHolder`. Esta forma de proceder mejora el rendimiento del `RecyclerView`, haciendo que funcione más rápido.

El método `onCreateViewHolder()` devuelve una vista de un elemento sin personalizar. Podríamos definir diferentes vistas para diferentes tipos de elementos utilizando el parámetro `viewType`. Usamos el método `inflate()` para crear una vista a partir del layout XML definido en `elemento_lista`. En este método se indica como segundo parámetro el layout padre que contendrá a la vista que se va a crear. En este caso, resulta imprescindible indicarlo, ya que queremos que la vista hijo ha de adaptarse al tamaño del padre (en `elemento_lista` se ha indicado `layout_width="match_parent"`). El tercer

parámetro del método permite indicar si queremos que la vista sea insertada en el padre. Indicamos `false`, dado que esta operación la va a hacer el `RecyclerView`.

El método `onBindViewHolder()` personaliza un elemento de tipo `ViewHolder` según su posición. A partir del `ViewHolder` que personalizamos ya es el sistema quien se encarga de crear la vista definitiva que será insertada en el `RecyclerView`. Finalmente, el método `getItemCount()` se utiliza para indicar el número de elementos a visualizar.

6. Ejecuta la aplicación y verifica el resultado.



Ejercicio: *Selección de un elemento en un RecyclerView*

En este ejercicio veremos cómo detectar que se ha pulsado sobre uno de los elementos del `RecyclerView`. En las vistas `ListView` y `GridView` podíamos realizar esta tarea usando el método `setOnItemClickListener()`. Sin embargo, en `RecyclerView` no se ha incluido este método. Google prefiere que asignemos un escuchador de forma independiente a cada una de las vistas que va a contener `RecyclerView`. Existen muchas alternativas para hacer este trabajo (En el capítulo 5 estudiaremos escuchadores de eventos y manejadores de eventos en Android). A continuación, explicamos una de ellas:

1. **En Java** Añade a la clase `MiAdaptador` la siguiente declaración:

```
protected View.OnClickListener onClickListener;
```

Para poder modificar el campo anterior añade el siguiente *setter*:

```
public void setOnItemClickListener(View.OnClickListener onClickListener) {
    this.onClickListener = onClickListener;
}
```

En Kotlin añade una nueva propiedad en el constructor principal:

```
class MiAdaptador(val lista: List<String>,
                  val onClick: (View) -> Unit) : …
```

2. Solo nos queda aplicar este escuchador a cada una de las vistas creadas. Añade la línea subrayada en el método indicado:

```
@Override
public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
    View v = inflador.inflate(R.layout.elemento_lista, parent, false);
    v.setOnClickListener(onClickListener);
    return new ViewHolder(v);
}
```

```
fun personaliza(lugar: Lugar, onClick: (View) -> Unit) = with(itemView) {
    …
    setOnClickListener{ onClick(itemView) }
}
```

3. Desde la clase `Puntuaciones` vamos a asignar un escuchador. Para ello añade el siguiente código al método `onCreate()`:

```
adaptador.setOnItemClickListener(new View.OnClickListener() {
    @Override public void onClick(View v) {
        int pos = recyclerView.getChildAdapterPosition(v);
        String s =  MainActivity.almacen.listaPuntuaciones(10).get(pos);
        Toast.makeText(Puntuaciones.this, "Selección: " + pos
              + " - " + s, Toast.LENGTH_LONG).show();
    }
});
```

```
adaptador = AdaptadorLugares((application as Aplicacion).lugares) {
   val pos: Int = recyclerView.getChildAdapterPosition(it)
   Toast.makeText(this, "pulsado $pos", Toast.LENGTH_LONG).show()
}
```

El método `getChildAdapterPosition()`, nos indicarán la posición de una vista dentro del adaptador.

4. Ejecuta la aplicación y verifica el resultado.

En los siguientes ejercicios usaremos un `RecyclerView` en Mis Lugares. La actividad inicial de la aplicación nos permite escoger entre cuatro botones. Sin embargo, sería mucho más interesante que en esta actividad se visualizara directamente una lista con los lugares.





Ejercicio: *Un RecyclerView en Mis Lugares*

1. Añade al fichero Gradle *Scripts/Bulid.gradle (Module:app)* la dependencia:

```
dependencies {
   …
   implementation 'androidx.recyclerview:recyclerview:1.1.0'
}
```

NOTA: Si lo deseas, puedes saltarte este paso. Más adelante, cuando aparezca RecyclerView en el código Java, la clase aparecerá marcada en rojo, al no encontrar su declaración. Al pulsar sobre la clase, aparecerá una bombilla roja donde podrás elegir la opción Add dependency on androidx.recyclerview:recyclerview. El mismo añadirá la dependencia, con la ventaja de seleccionar la última versión disponible.

La clase `RecyclerView` no ha sido añadida al API de Android, sino que se encuentra en una librería externa. Esto tiene la ventaja de que, aunque esta clase aparece en la versión 5 de Android, puede ser usada en versiones anteriores. Realmente, solo puede ser utilizada a partir del nivel de API 7. Pero el 100 % de los dispositivos que se comercializan en la actualidad cumplen este requisito.

2. Reemplaza el *layout content_main.xml* por el siguiente código:

```
<androidx.recyclerview.widget.RecyclerView
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:app="http://schemas.android.com/apk/res-auto"
android:id="@+id/recyclerView"
android:layout_width="match_parent"
android:layout_height="match_parent"
app:layout_behavior="@string/appbar_scrolling_view_behavior"
/>
```

3. En la práctica "Recursos alternativos en Mis Lugares" se crea un recurso alternativo para este layout en *res/layout-land/content_main.xml*. Elimina este recurso alternativo. ***NOTA:*** *Al borrarlo has de desactivar la opción Safe delete.*

4. En la actividad `MainActivity` añade el código subrayado:

```
public class MainActivity extends AppCompatActivity {
   …
   private RecyclerView recyclerView;
   public AdaptadorLugares adaptador;

   @Override protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      …
      adaptador = ((Aplicacion) getApplication()).adaptador;
      recyclerView = findViewById(R.id.recyclerView);
      recyclerView.setHasFixedSize(true);
      recyclerView.setLayoutManager(new LinearLayoutManager(this));
      recyclerView.setAdapter(adaptador);
   }
…
```

```
class MainActivity : AppCompatActivity() {
  …
  val lugares by lazy { (application as Aplicacion).lugares }
  val adaptador by lazy { (application as Aplicacion).adaptador }
  val usoLugar by lazy { CasosUsoLugar(this, lugares, adaptador) }

  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    …
```

```
        recyclerView.apply {
            setHasFixedSize(true)
            layoutManager = LinearLayoutManager(this@MainActivity)
            adapter = adaptador
        }
    }
…
```

En Java declaramos `recyclerView` y lo inicializamos con `findViewById()`. **En Kotlin** se hace automáticamente al estar en el Layout. Creamos un adaptador y se lo asignamos al `RecyclerView`. La clase `AdaptadorLugar` será definida a continuación. Además, indicamos que las vistas a mostrar serán de tamaño fijo y que usaremos un `LayoutManager` de tipo `LinearLayoutManager`.

5. De ser necesario, elimina del método `onCreate()` el código destinado a inicializar los botones. Los botones van a ser reemplazados por el `RecyclerView`.

6. En la clase `Aplicacion` crea la variable `adaptador`:

```
public AdaptadorLugares adaptador = new AdaptadorLugares(lugares);
```

```
val adaptador = AdaptadorLugares(lugares)
```

7. Ahora hemos de definir el layout que representará cada uno de los elementos de la lista. Crea el fichero *res/layout/elemento_lista.xml* con el siguiente código:

```
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="4dp">
    <ImageView android:id="@+id/foto"
            android:layout_width="?android:attr/listPreferredItemHeight"
            android:layout_height="?android:attr/listPreferredItemHeight"
            android:contentDescription="fotografía"
            android:src="@drawable/bar"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintLeft_toLeftOf="parent"/>
    <TextView android:id="@+id/nombre"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:text="Nombres del lugar"
            android:textAppearance="?android:attr/textAppearanceMedium"
            android:textStyle="bold"
            android:maxLines="1"
            app:layout_constraintTop_toTopOf="parent"
            app:layout_constraintStart_toEndOf="@+id/foto"
            app:layout_constraintEnd_toEndOf="parent"/>
    <TextView android:id="@+id/direccion"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:maxLines="1"
            android:text="dirección del lugar"
            app:layout_constraintTop_toBottomOf="@id/nombre"
            app:layout_constraintStart_toEndOf="@+id/foto"
            app:layout_constraintEnd_toEndOf="parent"/>
    <RatingBar android:id="@+id/valoracion"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            style="?android:attr/ratingBarStyleSmall"
            android:isIndicator="true"
            android:rating="3"
            app:layout_constraintTop_toBottomOf="@id/direccion"
            app:layout_constraintLeft_toRightOf="@+id/foto"
            app:layout_constraintBottom_toBottomOf="parent"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

Para combinar las vistas se ha escogido un `ConstraintLayout`. El primer elemento que contiene es un `ImageView` alineado a la izquierda. Su altura se establece a partir de un parámetro de configuración del sistema `?android:attr/listPreferredItemHeight` (altura preferida para ítem de lista). Su altura es la misma, por lo tanto, la imagen será cuadrada. A la derecha se muestran dos textos. En el texto de mayor tamaño se visualizará el nombre del lugar y en el de menor tamaño, la dirección. Bajo estos textos se ha incluido un `RatingBar`.

8. El siguiente paso será crear la clase *AdaptadorLugares*, que se encargará de rellenar el `RecyclerView`:

```
public class AdaptadorLugares extends
                    RecyclerView.Adapter<AdaptadorLugares.ViewHolder> {
    protected RepositorioLugares lugares;     // Lista de lugares a mostrar
    public AdaptadorLugares(RepositorioLugares lugares) {
        this.lugares = lugares;
    }
    //Creamos nuestro ViewHolder, con los tipos de elementos a modificar
    public static class ViewHolder extends RecyclerView.ViewHolder {
        public TextView nombre, direccion;
        public ImageView foto;
        public RatingBar valoracion;
        public ViewHolder(View itemView) {
            super(itemView);
            nombre = itemView.findViewById(R.id.nombre);
            direccion = itemView.findViewById(R.id.direccion);
            foto = itemView.findViewById(R.id.foto);
            valoracion= itemView.findViewById(R.id.valoracion);
        }
        // Personalizamos un ViewHolder a partir de un lugar
        public void personaliza(Lugar lugar) {
            nombre.setText(lugar.getNombre());
            direccion.setText(lugar.getDireccion());
            int id = R.drawable.otros;
            switch(lugar.getTipo()) {
                case RESTAURANTE:id = R.drawable.restaurante; break;
                case BAR:     id = R.drawable.bar;      break;
                case COPAS:    id = R.drawable.copas;    break;
```

```
            case ESPECTACULO:id = R.drawable.espectaculos; break;
            case HOTEL:   id = R.drawable.hotel;    break;
            case COMPRAS:  id = R.drawable.compras;   break;
            case EDUCACION: id = R.drawable.educacion; break;
            case DEPORTE:  id = R.drawable.deporte;   break;
            case NATURALEZA: id = R.drawable.naturaleza; break;
            case GASOLINERA: id = R.drawable.gasolinera; break;  }
         foto.setImageResource(id);
         foto.setScaleType(ImageView.ScaleType.FIT_END);
         valoracion.setRating(lugar.getValoracion());
      }
   }
   // Creamos el ViewHolder con la vista de un elemento sin personalizar
   @Override
   public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
      // Inflamos la vista desde el xml
      View v = LayoutInflater.from(parent.getContext())
                  .inflate(R.layout.elemento_lista, parent, false);
      return new ViewHolder(v);
   }
   // Usando como base el ViewHolder y lo personalizamos
   @Override
   public void onBindViewHolder(ViewHolder holder, int posicion) {
      Lugar lugar = lugares.elemento(posicion);
      holder.personaliza(lugar);
   }
   // Indicamos el número de elementos de la lista
   @Override public int getItemCount() {
      return lugares.tamaño();
   }
}
```

```
class AdaptadorLugares(val lugares: RepositorioLugares) :
                  RecyclerView.Adapter<AdaptadorLugares.ViewHolder>() {

  class ViewHolder(val view: View) : RecyclerView.ViewHolder(view) {

    fun personaliza(lugar: Lugar) = with(itemView){
      nombre.text = lugar.nombre
      direccion.text = lugar.direccion
      foto.setImageResource(when (lugar.tipoLugar) {
        TipoLugar.RESTAURANTE -> R.drawable.restaurante
        TipoLugar.BAR -> R.drawable.bar
        TipoLugar.COPAS -> R.drawable.copas
        TipoLugar.ESPECTACULO -> R.drawable.espectaculos
        TipoLugar.HOTEL -> R.drawable.hotel
        TipoLugar.COMPRAS -> R.drawable.compras
        TipoLugar.EDUCACION -> R.drawable.educacion
        TipoLugar.DEPORTE -> R.drawable.deporte
        TipoLugar.NATURALEZA -> R.drawable.naturaleza
        TipoLugar.GASOLINERA -> R.drawable.gasolinera
        TipoLugar.OTROS -> R.drawable.otros
      })
```



```
      foto.setScaleType(ImageView.ScaleType.FIT_END)
      valoracion.rating = lugar.valoracion
    }
  }

  override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):
                                                         ViewHolder {
    val v = LayoutInflater.from(parent.context)
                   .inflate(R.layout.elemento_lista, parent, false)
    return ViewHolder(v)
  }

  override fun onBindViewHolder(holder: ViewHolder, posicion: Int) {
    val lugar = lugares.elemento(posicion)
    holder.personaliza(lugar)
  }

  override fun getItemCount() = lugares.tamaño()
}
```

Un adaptador es un mecanismo estándar en Android que nos permite crear una serie de vistas que han de ser mostradas dentro de un contenedor. Con `RecyclerView` has de heredar de la clase `RecyclerView.Adapter` para crear su adaptador.

En el constructor se inicializa el conjunto de datos a mostrar (en el ejemplo `lugares`). Luego se crea la clase `ViewHolder`, que contendrá las vistas que queremos modificar de un elemento (en concreto: dos `TextView` con el nombre y la dirección, un `ImageView` con la imagen del tipo de lugar y un `RatingBar`). En Kotlin estas cuatro vistas no son declaradas de forma explícita, se realiza de forma automática (ver `import kotlinx…elemento_lista.view.*`). La clase `ViewHolder` es utilizada para evitar tener que crear las vistas de cada elemento desde cero. Lo va a hacer es utilizar un `ViewHolder` que contendrá las cuatro vistas ya creadas, pero sin personalizar. De forma que, gastará el mismo `ViewHolder` para todos los elementos y simplemente lo personalizaremos según la posición. Es decir, reciclamos el `ViewHolder`. Esta forma de proceder mejora el rendimiento del `RecyclerView`, haciendo que funcione más rápido.

El método `onCreateViewHolder()` devuelve una vista de un elemento sin personalizar. Podríamos definir diferentes vistas para diferentes tipos de elementos utilizando el parámetro `viewType`. Usamos el método `inflate()` para crear una vista a partir del layout XML definido en `elemento_lista`. En este método se indica como segundo parámetro el layout padre que contendrá a la vista que se va a crear. En este caso, resulta imprescindible indicarlo, ya que queremos que la vista hijo ha de adaptarse al tamaño del padre (en `elemento_lista` se ha indicado `layout_width="match_parent"`). El tercer parámetro del método permite indicar si queremos que la vista sea insertada en el padre. Indicamos `false`, dado que esta operación la va a hacer el `RecyclerView`.

El método `onBindViewHolder()` personaliza un elemento de tipo `ViewHolder` según su posición. A partir del `ViewHolder` que personalizamos ya es el sistema quien se encarga de crear la vista definitiva que será insertada en el

RecyclerView. Finalmente, el método `getItemCount()` se utiliza para indicar el número de elementos a visualizar.

9. Ejecuta la aplicación y verifica el resultado.



Práctica: *Selección de un elemento en un RecyclerView* en *Mis Lugares*

Queremos que cuando se seleccione un elemento de la actividad principal de Mis Lugares, se abra la actividad que visualiza su contenido. Puedes basarte en el ejercicio *Selección de un elemento en un RecyclerView*. Una solución a esta práctica la encontrarás al final de http://www.androidcurso.com/index.php/691.



Preguntas de repaso: *RecyclerView*

3.11. Las intenciones

Una intención representa la voluntad de realizar alguna acción o tarea, como realizar una llamada de teléfono o visualizar una página web. Una intención nos permite lanzar una actividad o servicio de nuestra aplicación o de una aplicación diferente. Tienen un gran potencial en Android, por lo que resulta importante conocerlas y dominarlas.



Vídeo[tutorial]: *Las intenciones en Android*

Existen dos tipos de intenciones:

- **Intenciones explícitas:** se indica exactamente el componente a lanzar. Su utilización típica es la de ir ejecutando los diferentes componentes internos de una aplicación. Por ejemplo, desde Asteroides o Mis Lugares lanzamos `AcercaDeActivity` por medio de una intención explicita.
- **Intenciones implícitas:** pueden solicitar tareas abstractas, como "quiero tomar una foto" o "quiero enviar un mensaje". Además, las intenciones se resuelven en tiempo de ejecución, de forma que el sistema mirará cuántos componentes han registrado la posibilidad de ejecutar ese tipo de intención. Si encuentra varias, el sistema puede preguntar al usuario el componente que prefiere utilizar.

Además, como se ha estudiado en el apartado "Comunicación entre actividades", las intenciones ofrecen un servicio de paso de mensajes que permite interconectar datos entre componentes.

En concreto, se utilizan intenciones cada vez que queramos:

- Lanzar una *actividad* (`startActivity()` y `startActivityForResult()`).
- Lanzar un *servicio* (`startService()`).

- Lanzar un *anuncio de tipo broadcast* (`sendBroadcast()`).
- Conectamos con un *servicio* (`bindService()`).

En muchas ocasiones, una intención no será inicializada por la aplicación, si no por el sistema; por ejemplo, cuando pedimos visualizar una página web. En otras ocasiones será necesario que la aplicación inicialice su propia intención. Para ello se creará un objeto de la clase `Intent`.

Cuando se crea una intención (es decir, se instancia un objeto de tipo `Intent`), esta contiene información de interés para que el sistema trate adecuadamente la intención o para el componente que recibe la intención. Puede incluir la siguiente información:

Nombre del componente: Identificamos el componente que queremos lanzar con la intención. Hay que utilizar el nombre de clase totalmente cualificado que queremos lanzar (`org.example.asteroides.AcercaDeActivity`). El nombre del componente es opcional. En caso de no indicarse, se utilizará otra información de la intención para obtener el componente a lanzar. A este tipo de intenciones se las conocía como intenciones explícitas.

Acción: Una cadena de caracteres donde indicamos la acción a realizar o, en caso de un receptor de anuncios (*Broadcast Receiver*), la acción que tuvo lugar y que queremos reportar. La clase `Intent` define una serie de constantes para acciones genéricas que se enumeran a continuación:

Constante	Acción
ACTION_VIEW	Visualiza un contacto, mapa, página Web,... o reproduce música o vídeo,
ACTION_INSERT	Inserta un contacto o cita en calendario
ACTION_EDIT	Edita un contacto o cita en calendario
ACTION_MAIN	Arranca como actividad principal de una tarea
ACTION_CALL	Inicializa una llamada de teléfono
ACTION_DIAL	Introduce un número sin llegar a realizar la llamada
ACTION_SEND	Manda un correo o SMS
ACTION_SET_TIMER	Programa un temporizados
ACTION_SET_ALARM	Programa una alarma.
ACTION_IMAGE_CAPTURE	Tomar una foto
ACTION_VIDEO_CAPTURE	Grabar un vídeo
ACTION_GET_CONTENT	Seleccionar un tipo de archivo específico
ACTION_OPEN_DOCUMENT	Abrir un tipo de archivo específico
ACTION_CREATE_DOCUMENT	Crear un tipo de archivo específico
ACTION_PICK	Seleccionar un contacto o fichero

Tabla 5: *Algunas acciones estándar de las intenciones.*

También puedes definir tus propias acciones. En ese caso has de *indicar* el paquete de tu aplicación como prefijo. Por ejemplo: `org.example.asteroides.MUESTRA_PUNTUACIONES`.

Categoría: Complementa a la acción. Indica información adicional sobre el tipo de componente que ha de ser lanzado. El número de categorías puede ampliarse arbitrariamente. No obstante, en la clase `Intent` se definen una serie de categorías genéricas que podemos utilizar.

Constante	Significado
CATEGORY_BROWSABLE	La actividad lanzada puede ser invocada con seguridad por el navegador para mostrar los datos referenciados por un enlace (por ejemplo, una imagen o un mensaje de correo electrónico).
CATEGORY_HOME	La actividad muestra la pantalla de inicio, la primera pantalla que ve el usuario cuando el dispositivo está encendido o cuando se presiona la tecla *Home*.
CATEGORY_LAUNCHER	La actividad puede ser la actividad inicial de una tarea y se muestra en el lanzador de aplicaciones de nivel superior.
CATEGORY_PREFERENCE	La actividad a lanzar es un panel de preferencias.

***Tabla 6:** Algunas categorías estándar de las intenciones.*

Una categoría suele utilizarse junto con una acción para aportar información adicional. Así, por ejemplo, indicaremos `ACTION_MAIN` a las actividades que pueden utilizarse como puntos de entrada de una aplicación. Indicaremos, además, `CATEGORY_LAUNCHER` para que la actividad se muestre en la pantalla de inicio.

Datos: Referencia a los datos con los que trabajaremos. Hay que expresar estos datos por medio de una URI (el mismo concepto ampliamente utilizado en Internet). Ejemplos de URI son: tel:963228525, http://www.androidcurso.com, content://call_log/calls... En muchos casos, resulta importante saber el tipo de datos con el que se trabaja. Con este propósito se indica el tipo MIME asociado a la URI, es decir, se utiliza el mismo mecanismo que en Internet. Ejemplos de tipos MIME son: `text/xml`, `image/jpeg`, `audio/mp3`...

Extras: Información adicional que será recibida por el componente lanzado. Está formada por un conjunto de pares variable/valor. Estas colecciones de valores se almacenan en un objeto de la clase `Bundle`. Su utilización se ha descrito en el apartado "Comunicación entre actividades". Recordemos cómo se introducían estos valores en un `Intent`.

```
intent.putExtra("usuario", "Pepito Perez");
intent.putExtra("edad", 27);
```

En el apartado "Creación de nuevas actividades" hemos aprendido a lanzar una actividad de forma explícita utilizando el constructor `Intent(Context contexto, Class<?> clase)`. Por ejemplo, para lanzar la actividad `AcercaDeActivity` escribíamos:


```
Intent intent = new Intent(this, AcercaDeActivity.class);
startActivity(intent);
```

Para lanzar una actividad de forma implícita podemos usar el constructor `Intent(String action, Uri uri)`. Por ejemplo:

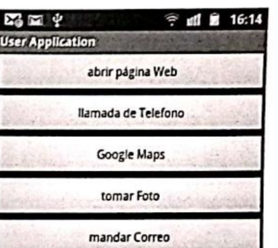
```
Intent intent = new Intent(Intent.ACTION_DIAL,
                           URI.parse("tel:962849347");
startActivity(intent);
```

También se puede utilizar `startActivityForResult()` si esperamos que la actividad nos devuelva datos.



Ejercicio: *Uso de intenciones implícitas*

1. Crea un nuevo proyecto con nombre *Intenciones* y tipo *Empty Activity.*
2. El *layout* de la actividad inicial ha de estar formado por cinco botones, tal y como se muestra a continuación:



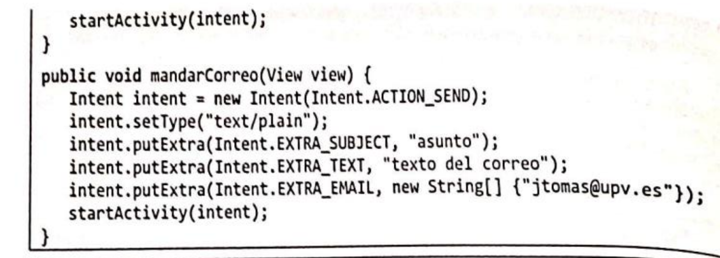
3. Abre la actividad principal e incorpora los siguientes métodos:

```
public void verPgWeb(View view) {
  Intent intent = new Intent(Intent.ACTION_VIEW,
                        Uri.parse("http://www.androidcurso.com/"));
  startActivity(intent);
}

public void llamarTelefono(View view) {
  Intent intent = new Intent(Intent.ACTION_DIAL,
                        Uri.parse("tel:962849347"));
  startActivity(intent);
}

public void googleMaps(View view) {
  Intent intent = new Intent(Intent.ACTION_VIEW,
                        Uri.parse("geo:41.656313,-0.877351"));
  startActivity(intent);
}

public void tomarFoto(View view) {
  Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
```



```
    startActivity(intent);
}
public void mandarCorreo(View view) {
    Intent intent = new Intent(Intent.ACTION_SEND);
    intent.setType("text/plain");
    intent.putExtra(Intent.EXTRA_SUBJECT, "asunto");
    intent.putExtra(Intent.EXTRA_TEXT, "texto del correo");
    intent.putExtra(Intent.EXTRA_EMAIL, new String[] {"jtomas@upv.es"});
    startActivity(intent);
}
```

```
fun verPgWeb(view: View) = startActivity(
    Intent(Intent.ACTION_VIEW, Uri.parse("http://www.androidcurso.com/")))

fun llamarTelefono(view: View) = startActivity(
    Intent(Intent.ACTION_DIAL, Uri.parse("tel:962849347")))

fun googleMaps(view: View) = startActivity(
    Intent(Intent.ACTION_VIEW, Uri.parse("geo:41.656313,-0.877351")))

fun tomarFoto(view: View) = startActivity(
    Intent(MediaStore.ACTION_IMAGE_CAPTURE))

fun mandarCorreo(view: View) = startActivity(
    Intent(Intent.ACTION_SEND).apply {
        type = "text/plain"
        putExtra(Intent.EXTRA_SUBJECT, "asunto")
        putExtra(Intent.EXTRA_TEXT, "texto del correo")
        putExtra(Intent.EXTRA_EMAIL, arrayOf("jtomas@upv.es"))})
```

4. Asocia el atributo `onClick` de cada uno de los botones al método correspondiente.
5. Ejecuta la aplicación en un terminal real. Observa como el botón de mandar correo te permite seleccionar entre diferentes aplicaciones con esta funcionalidad.



6. Este resultado puede variar en función de las aplicaciones instaladas.



Recursos adicionales: *Tabla con intenciones que podemos utilizar de aplicaciones Google*

Aplicación	URI	Acción	Resultado
Navegador	http://*dirección_web* https://*dirección_web*	VIEW	Abre una ventana de navegador con una URL.
	"" (cadena vacía) http://*dirección_web* https://*dirección_web*	WEB_SEARCH	Realiza una búsqueda web. Se indica la cadena de búsqueda en el extra `SearchManager.QUERY`
Teléfono	tel:*número_teléfono*	CALL	Realiza una llamada de teléfono. Los números válidos se definen en IETF RFC 3966 . Entre estos se incluyen: tel:2125551212 y tel:(212)5551212. Necesitamos el permiso `android.permission.CALL_PHONE`
	tel:*número_teléfono* *voicemail:*	DIAL	Introduce un número sin llegar a realizar la llamada.
Google Maps	geo:*latitud,longitud* geo:*lat,long*?z=*zoom* geo:0,0?q=*dirección* geo:0,0?q=*búsqueda*	VIEW	Abre la aplicación Google Maps para una localización determinada. El campo *z* especifica el nivel de zum.
Google Streetview	google.streetview: cbll=*latitud,longitud*& cbp=1,*yaw,pitch,zoom* & mz=*mapZoom*	VIEW	Abre la aplicación Street View para la ubicación dada. El esquema de URI se basa en la sintaxis que utiliza Google Maps. Solo el campo *cbll* es obligatorio.

Puedes consultar una lista con las intenciones comunes en: https://developer.android.com/guide/components/intents-common?hl=es-419



Práctica: *Uso de intenciones implícitas*

1. Crea nuevos botones en la aplicación del ejercicio anterior y experimenta con otro tipo de acciones y URI. Puedes consultar la tabla anterior. A continuación, tienes algunas propuestas:
2. Compara las acciones `VIEW` y `WEB_SEARCH`. ¿Encuentras alguna diferencia?
3. Compara las acciones `CALL` y `DIAL`. ¿Encuentras alguna diferencia?
4. Experimenta con Google Streetview.



Ejercicio: *Intenciones implícitas en Mis Lugares*

1. Si has hecho el ejercicio *Casos de Uso para lugares*, en la clase `CasosUsoLugar` añade cuatro nuevos casos de uso: Si no añádelos a la clase `VistaLugarActivity`.

```
// INTENCIONES
public void compartir(Lugar lugar) {
   Intent i = new Intent(Intent.ACTION_SEND);
   i.setType("text/plain");
   i.putExtra(Intent.EXTRA_TEXT,
              lugar.getNombre() + " - " + lugar.getUrl());
   actividad.startActivity(i);
}

public void llamarTelefono(Lugar lugar) {
   actividad.startActivity(new Intent(Intent.ACTION_DIAL,
            Uri.parse("tel:" + lugar.getTelefono())));
}

public void verPgWeb(Lugar lugar) {
   actividad.startActivity(new Intent(Intent.ACTION_VIEW,
            Uri.parse(lugar.getUrl())));
}

public final void verMapa(Lugar lugar) {
   double lat = lugar.getPosicion().getLatitud();
   double lon = lugar.getPosicion().getLongitud();
   Uri uri = lugar.getPosicion() != GeoPunto.SIN_POSICION
      ? Uri.parse("geo:" + lat + ',' + lon)
      : Uri.parse("geo:0,0?q=" + lugar.getDireccion());
   actividad.startActivity(new Intent("android.intent.action.VIEW", uri));
}
```

```
// INTENCIONES
fun compartir(lugar: Lugar) = actividad.startActivity(
   Intent(Intent.ACTION_SEND).apply {
      type = "text/plain"
      putExtra(Intent.EXTRA_TEXT, "$(lugar.nombre) - $(lugar.url)")
   })

fun llamarTelefono(lugar: Lugar) = actividad.startActivity(
   Intent(Intent.ACTION_DIAL, Uri.parse("tel:" + lugar.telefono)))

fun verPgWeb(lugar: Lugar) = actividad.startActivity(
   Intent(Intent.ACTION_VIEW, Uri.parse(lugar.url)))

fun verMapa(lugar: Lugar) {
   val lat = lugar.posicion.latitud
   val lon = lugar.posicion.longitud
   val uri = if (lugar.posicion != GeoPunto.SIN_POSICION)
               Uri.parse("geo:$lat,$lon")
          else Uri.parse("geo:0,0?q=" + lugar.direccion)
   actividad.startActivity(Intent(Intent.ACTION_VIEW, uri))
}
```


El primer método crea una intención implícita con acción `ACTION_SEND`. Mandaremos un texto plano formado por el nombre del lugar y su URL. Esta información podrá ser recogida por cualquier aplicación que se haya registrado como enviadora de mensajes (WhatsApp, Gmail, SMS, etc.).

El segundo método realiza una llamada telefónica al número del lugar. El tercero abre su página Web. El cuarto obtiene la latitud y longitud del lugar. Si alguna de las dos es distinta de cero, consideraremos que se ha introducido esta información y crearemos una URI basada en estos valores. Si son cero, consideraremos que no se han introducido, por lo que crearemos una URI basándonos en la dirección del lugar.

2. En la clase `VistaLugarActivity` hay que llamar a dos de estos métodos desde el menú. En el método `onOptionsItemSelected()` añade el código subrayado:

```
@Override public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {
  case R.id.accion_compartir:
     usoLugar.compartir(lugar);
     return true;
  case R.id.accion_llegar:
     usoLugar.verMapa(lugar);
     return true;
     …
```

```
override fun onOptionsItemSelected(item: MenuItem): Boolean {
  when (item.getItemId()) {
     R.id.accion_compartir -> {
        usoLugar.compartir(lugar)
        return true
     }
     R.id.accion_llegar -> {
        usoLugar.verMapa(lugar)
        return true
     }
     …
```

3. Ejecuta la aplicación, selecciona alguno de los lugares y utiliza la barra de acciones para verificar estas opciones. Para la segunda opción, es importante que el terminal disponga de algún *software* de mapas (como Google Maps).

4. Abre el *layout vista_lugar.xml*. Localiza el `LinearLayout` que contiene el icono y el texto con la dirección. Añade el atributo `onClick` como se muestra a continuación:

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="horizontal"
  android:onClick="verMapa">
    <ImageView
        android:layout_width="40dp"
        android:layout_height="40dp"
        android:contentDescription="Logo de la dirección"
```

```
        android:src="@android:drawable/ic_menu_myplaces" />
    ...
```

5. Añade también el atributo `onClick` en los `LinearLayout` que contienen el teléfono y la URL utilizando el método adecuado.
6. Añade a `VistaLugarActivity` los siguientes métodos:

```
public void verMapa(View view) {
   usoLugar.verMapa(lugar);
}

public void llamarTelefono(View view) {
   usoLugar.llamarTelefono(lugar);
}

public void verPgWeb(View view) {
   usoLugar.verPgWeb(lugar);
}
```

```
fun verMapa(view: View) = usoLugar.verMapa(lugar)

fun llamarTelefono(view: View) = usoLugar.llamarTelefono(lugar)

fun verPgWeb(view: View) = usoLugar.verPgWeb(lugar)
```

7. Verifica el funcionamiento de las nuevas intenciones implícitas.

3.11.1. Añadiendo fotografías en Mis Lugares

En este apartado seguiremos trabajando con el uso de intenciones aplicándolas a la aplicación Mis Lugares. En concreto, permitiremos que el usuario pueda asignar fotografías a cada lugar utilizando ficheros almacenados en su dispositivo o la cámara.



Ejercicio: *Añadiendo fotografías desde la galería*

1. En la clase `VistaLugarActivity` añade las siguientes constantes y atributos:

```
final static int RESULTADO_GALERIA = 2;
final static int RESULTADO_FOTO = 3;
private ImageView foto;
```

```
val RESULTADO_GALERIA = 2
val RESULTADO_FOTO = 3
```

Desde la actividad `VistaLugarActivity` llamamos a diferentes actividades y algunas de ellas nos tienen que devolver información. En estos casos llamamos a la actividad con `startActivityForResult()` pasándole un código que identifica la llamada. Cuando esta actividad termine, se llamará al método `onActivityResult()`, que nos indicará el mismo código usado en la llamada. Como vamos a hacerlo con tres actividades diferentes, hemos creado tres constantes, con los respectivos códigos de respuesta. Actuando de esta forma conseguimos un código más legible.

2. **En Java**, en el método `onCreate()` añade antes de `actualizaVistas()`:

```
foto = findViewById(R.id.foto);
```

3. Busca en *vista_lugar.xml* el `ImageView` con descripción "logo galería" y añade el atributo:

```
android:onClick="ponerDeGaleria"
```

4. Añade la siguiente función en la actividad:

```
public void ponerDeGaleria(View view) {
   usoLugar.ponerDeGaleria(RESULTADO_GALERIA);
}
```

```
fun ponerDeGaleria(view: View)= usoLugar.ponerDeGaleria(RESULTADO_GALERIA)
```

5. Añade el siguiente caso de uso a la clase `CasosUsoLugar`:

```
// FOTOGRAFÍAS
public void ponerDeGaleria(int codidoSolicitud) {
   String action;
   if (android.os.Build.VERSION.SDK_INT >= 19) { // API 19 - Kitkat
      action = Intent.ACTION_OPEN_DOCUMENT;
   } else {
      action = Intent.ACTION_PICK;
   }
   Intent intent = new Intent(action,
                   MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
   intent.addCategory(Intent.CATEGORY_OPENABLE);
   intent.setType("image/*");
   actividad.startActivityForResult(intent, codidoSolicitud);
}
```

```
// FOTOGRAFÍAS
fun ponerDeGaleria(codidoSolicitud: Int) {
  val action = if (android.os.Build.VERSION.SDK_INT >= 19)  // Kitkat
                  Intent.ACTION_OPEN_DOCUMENT
               else Intent.ACTION_PICK
  val i=Intent(action,MediaStore.Images.Media.EXTERNAL_CONTENT_URI).apply {
    addCategory(Intent.CATEGORY_OPENABLE)
    type = "image/*"
  }
  actividad.startActivityForResult(i, codidoSolicitud)
}
```

Este método crea una intención indicando que queremos seleccionar contenido. El contenido será proporcionado por el *Content Provider* `MediaStore`; además le indicamos que nos interesan imágenes del almacenamiento externo. Típicamente se abrirá la aplicación galería de fotos (u otra similar). Observa que se usan dos acciones diferentes: `ACTION_OPEN_DOCUMENT` solo está disponible a partir del API 19 y tiene la ventaja que no requiere que la aplicación pida permiso de lectura. Cuando el usuario selecciona un fichero el *Content Provider*, dará a

nuestra aplicación permiso de lectura (o incluso de escritura) pero solo para el archivo solicitado. No se considera una acción peligrosa dado que es el usuario quien selecciona el archivo a compartir con la aplicación. Si se ejecuta en un API anterior al 19, tendremos que usar `ACTION_PICK`, que sí que requiere dar permisos de lectura en la memoria externa. Una vez concedido este permiso la aplicación podría aprovechar y leer otros ficheros sin la intervención directa del usuario. Como necesitamos una respuesta, usamos `startActivityForResult()` con el código adecuado.

6. Si la versión del dispositivo es anterior al API 19, vamos a tener que pedir permiso para leer ficheros de la memoria externa. En *AndroidManifest.xml* añade dentro de la etiqueta `<manifest …> </manifest>` el siguiente código:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

7. En el método `onActivityResult()` añade la sección `else if` que se muestra:

```
if (requestCode == RESULTADO_EDITAR) {
   …
} else if (requestCode == RESULTADO_GALERIA) {
   if (resultCode == Activity.RESULT_OK) {
      usoLugar.ponerFoto(pos, data.getDataString(), foto);
   } else {
      Toast.makeText(this, "Foto no cargada",Toast.LENGTH_LONG).show();
   }
}
```

```
if (requestCode == RESULTADO_EDITAR) {
   …
} else if (requestCode == RESULTADO_GALERIA) {
   if (resultCode == RESULT_OK) {
      usoLugar.ponerFoto(pos, data?.dataString ?: "", foto)
   } else {
      Toast.makeText(this, "Foto no cargada", Toast.LENGTH_LONG).show();
   }
}
```

Comenzamos verificando que volvemos de la actividad lanzada por la intención anterior. Comprobamos que el usuario no ha cancelado la operación. En este caso, nos tiene que haber pasado en la intención de respuesta, `data`, una URI con la foto seleccionada. Esta URI puede ser del tipo `file://…`, `content://…` o `http://…` según qué aplicación haya resuelto esta intención. El siguiente paso consiste en modificar el contenido de la vista que muestra la foto, `imageView`, con esta URI. Lo hacemos llamando al caso de uso `ponerFoto()`:

8. Añade en `CasosUsoLugar`:

```
public void ponerFoto(int pos, String uri, ImageView imageView) {
   Lugar lugar = lugares.elemento(pos);
   lugar.setFoto(uri);
   visualizarFoto(lugar, imageView);
}
```



```
public void visualizarFoto(Lugar lugar, ImageView imageView) {
   if (lugar.getFoto() != null && !lugar.getFoto().isEmpty()) {
      imageView.setImageURI(Uri.parse(lugar.getFoto()));
   } else {
      imageView.setImageBitmap(null);
   }
}
```

```
fun ponerFoto(pos: Int, uri: String?, imageView: ImageView) {
   val lugar = lugares.elemento(pos)
   lugar.foto = uri ?: ""
   visualizarFoto(lugar, imageView)
}
fun visualizarFoto(lugar: Lugar, imageView: ImageView) {
   if (!(lugar.foto == null || lugar.foto.isEmpty())) {
      imageView.setImageURI(Uri.parse(uri))
   } else {
      imageView.setImageBitmap(null)
   }
}
```

El primer método comienza obteniendo el lugar que corresponde al `id` para modificar la URI de su foto. Luego llama a `visualizarFoto()`. Este verifica que la URI que acabamos de asignar no está vacía. Si es así, la asigna al `ImageView` que nos han pasado para que la imagen se represente en pantalla. En caso contrario, se le asigna un `Bitmap` igual a `null`, que es equivalente a que no se represente ninguna imagen.

9. Ya puedes ejecutar la aplicación. Si añades una fotografía a un lugar, esta se visualizará. Sin embargo, si vuelve a la lista de lugares y seleccionas el mismo lugar al que asignaste la fotografía, ésta ya no se representa. La razón es que no hemos visualizado la foto al crear la actividad.

10. En el método `actualizaVistas()` añade la siguiente línea al final:

```
usoLugar.visualizarFoto(lugar, foto);
```

11. Verifica de nuevo el funcionamiento de la aplicación.

Ejercicio: *Añadiendo fotografías desde la cámara*

1. Añade el siguiente atributo a la clase `VistaLugarActivity`:

```
private Uri uriUltimaFoto;
```

```
private var uriUltimaFoto: Uri? = null
```

Como veremos, necesitamos esta variable en dos métodos diferentes. Por lo tanto, la declaramos de forma global.

2. Añade el siguiente caso de uso a la clase `CasosUsoLugar`:

```
public Uri tomarFoto(int codidoSolicitud) {
   try {
      Uri uriUltimaFoto;
      File file = File.createTempFile(
            "img_" + (System.currentTimeMillis()/ 1000), ".jpg" ,
            actividad.getExternalFilesDir(Environment.DIRECTORY_PICTURES));
      if (Build.VERSION.SDK_INT >= 24) {
         uriUltimaFoto = FileProvider.getUriForFile(
               actividad, "es.upv.jtomas.mislugares.fileProvider", file);
      } else {
         uriUltimaFoto = Uri.fromFile(file);
      }
      Intent intent   = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
      intent.putExtra (MediaStore.EXTRA_OUTPUT, uriUltimaFoto);
      actividad.startActivityForResult(intent, codidoSolicitud);
      return uriUltimaFoto;
   } catch (IOException ex) {
      Toast.makeText(actividad, "Error al crear fichero de imagen",
            Toast.LENGTH_LONG).show();
      return null;
   }
}
```

```
fun tomarFoto(codidoSolicitud: Int): Uri? {
   try {
      val file = File.createTempFile(
         "img_" + System.currentTimeMillis() / 1000, ".jpg",
         actividad.getExternalFilesDir(Environment.DIRECTORY_PICTURES) )
      val uriUltimaFoto = if (Build.VERSION.SDK_INT >= 24)
         FileProvider.getUriForFile(
            actividad, "es.upv.jtomas.mislugares.fileProvider", file )
      else Uri.fromFile(file)
      val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
      intent.putExtra(MediaStore.EXTRA_OUTPUT, uriUltimaFoto)
      actividad.startActivityForResult(intent, codidoSolicitud)
      return uriUltimaFoto
   } catch (ex: IOException) {
      Toast.makeText(actividad, "Error al crear fichero de imagen",
                  Toast.LENGTH_LONG).show()
      return null
   }
}
```

Este método crea una intención indicando que queremos capturar una imagen desde el dispositivo. Típicamente se abrirá la aplicación cámara de fotos. A esta intención vamos a añadirle un extra con una URI al fichero donde queremos que se almacene la fotografía. Para crear el fichero, se utiliza `createTempFile()` indicando nombre, extensión y directorio. El método `currentTimeMillis()` nos da el número de milisegundos transcurridos desde 1970. Al dividir entre 1000, tenemos el número de segundos. El objetivo que se persigue es que, al crear un nuevo fichero, su nombre nunca coincida con uno anterior. El directorio del fichero será el utilizado para almacenar fotos privadas en la memoria externa. Estos ficheros serán de uso exclusivo para tu aplicación. Además, si desinstalas

la aplicación, este directorio será borrado. Si quieres que los ficheros sean de acceso público utiliza `getExternalStoragePublicDirectory()`.

Una vez tenemos el fichero hay dos alternativas para crear la URI. Si el API de Android donde se ejecuta la aplicación es 24 o superior, podemos crear el fichero asociado a un *Content Provider* nuestro. Esta acción no requiere solicitar permiso de escritura. Si el API es anterior al 24, no se dispone de esta acción, y el fichero será creado de forma convencional en la memoria externa. El inconveniente es que para realizar esta acción tendremos que pedir al usuario permiso de escritura en la memoria externa. Al concedernos este permiso, también podremos borrar o sobreescribir cualquier fichero que el usuario tenga en esta memoria. Por lo que muchos usuarios no querrán darnos este permiso. Al final, llamamos a `startActivityForResult()` con el código que nos han pasado.

3. Si la versión mínima de API es anterior a 24, vamos a tener que pedir permiso para leer ficheros de la memoria externa. En *AndroidManifest.xml* añade dentro de la etiqueta `<manifest …> </manifest>` el siguiente código:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

NOTA: Observa cómo hemos tenido que solicitar permiso para acceder a la memoria externa, pero no es necesario solicitar permiso para tomar una fotografía. La razón es que realmente nuestra aplicación no toma la fotografía directamente, si no que por medio de una intención lanzamos otra aplicación, que sí que tiene este permiso.

4. En un punto anterior hemos utilizado un Content Provider para almacenar ficheros. Para crearlo añade en *AndroidManifest.xml* dentro de la etiqueta `<application …> </application>` el siguiente código:

```
<provider
   android:name="androidx.core.content.FileProvider"
   android:authorities="es.upv.jtomas.mislugares.fileProvider"
   android:exported="false"
   android:grantUriPermissions="true">
   <meta-data  android:name="android.support.FILE_PROVIDER_PATHS"
               android:resource="@xml/file_paths" />
</provider>
```

5. Crea un nuevo recurso con nombre "file_paths.xml" en la carpeta res/ xml:

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
   <external-files-path name="my_images" path="/" />
</paths>
```

Has de cambiar `es.upv.jtomas…` por un identificador unido que incluya tu nombre o empresa. Ha de coincidir con el valor indicado en la función `tomarFoto()`. Cambia `com.example.package.name` por el paquete de la aplicación.

6. Busca en *vista_lugar.xml* el `ImageView` con descripción "logo cámara" y añade el atributo:

```
android:onClick="tomarFoto"
```

7. En `VistaLugarActivity` añade:

```
public void tomarFoto(View view) {
   uriUltimaFoto = usoLugar.tomarFoto(RESULTADO_FOTO);
}
```

```
fun tomarFoto(view: View) {
   uriUltimaFoto = usoLugar.tomarFoto(RESULTADO_FOTO)
}
```

8. En el método `onActivityResult()` añade la sección `else if` que se muestra:

```
} else if (requestCode == RESULTADO_GALERIA
   …
} else if (requestCode == RESULTADO_FOTO) {
   if (resultCode == Activity.RESULT_OK && uriUltimaFoto!=null) {
      lugar.setFoto(uriUltimaFoto.toString());
      usoLugar.ponerFoto(pos, lugar.getFoto(), foto);
   } else {
      Toast.makeText(this, "Error en captura", Toast.LENGTH_LONG).show();
   }
}
```

```
} else if (requestCode == RESULTADO_GALERIA) {
   …
} else if (requestCode == RESULTADO_FOTO) {
   if (resultCode == Activity.RESULT_OK && uriUltimaFoto!=null) {
      lugar.foto = uriUltimaFoto.toString()
      usoLugar.ponerFoto(pos, lugar.foto, foto);
   } else {
      Toast.makeText(this, "Error en captura", Toast.LENGTH_LONG).show()
   }
}
```

Comenzamos verificando que volvemos de la actividad lanzada por la intención anterior, que el usuario no ha cancelado la operación y que `uriUltimaFoto` ha sido inicializado. En este caso, se nos pasa información en la intención de respuesta, pero sabemos que en `uriUltimaFoto` está almacenada la URI con el fichero donde se ha almacenado la foto. Guardamos esta URI en el campo adecuado de `lugar` e indicamos que se guarde y se represente en la vista `foto`.

9. Verifica de nuevo el funcionamiento de la aplicación.

NOTA: En algunos dispositivos puede aparecer un error de memoria si la cámara está configurada con mucha resolución. Entonces puedes probar con la cámara delantera.



Ejercicio: *Añadiendo un botón para eliminar fotografías*

1. En el *layout vista_lugar.xml* añade el siguiente botón dentro del `LinearLayout` donde están los botones para la cámara y para la galería:

```
<ImageView
   android:layout_width="40dp"
   android:layout_height="40dp"
   android:contentDescription="Eliminar foto"
   android:onClick="eliminarFoto"
   android:src="@android:drawable/ic_menu_close_clear_cancel" />
```

2. Añade el siguiente método a la clase `VistaLugarActivity`:

```
public void eliminarFoto(View view) {
   usoLugar.ponerFoto(pos, "", foto);
}
```

```
fun eliminarFoto(view: View) = usoLugar.ponerFoto(pos, "", foto)
```

3. Verifica el funcionamiento del nuevo botón.

NOTA: Las fotografías introducidas por el usuario pueden tener muy diversas procedencias, pudiendo tener grandes tamaños. Cuando trabajas con fotografías es muy importante que tengas en cuenta que la memoria es un recurso limitado. Por lo tanto, es muy probable que cuando trates de cargar una imagen demasiado grande, tu aplicación se detenga, mostrando en el LogCat un error de memoria. Para resolver el problema se podría cargar la imagen a una resolución menor, adecuada para un dispositivo móvil. Para ello puedes utilizar una librería de carga de imágenes como Glide o Picasso. También puedes realizar este proceso siguiendo esta documentación [21].



Práctica: *Confirmar borrado fotografías*

Si lo deseas, puedes poner un cuadro de diálogo para confirmar la eliminación. Puedes basarte en la práctica «Un cuadro de diálogo para confirmar el borrado».



Enlaces de interés: *Aprender más sobre intenciones*

- *Android Developers - Reference: Class Intent:*
  http://developer.android.com/reference/android/content/Intent.html
- *Android Developers – Dev. Guide: Intents and Intent Filters:*
  http://developer.android.com/guide/topics/intents/intents-filters.html



Preguntas de repaso: *Intenciones*

[21] https://developer.android.com/topic/performance/graphics/load-bitmap

CAPÍTULO 4.
Gráficos en Android

Android nos proporciona a través de su API gráfica una potente y variada colección de funciones que pueden cubrir prácticamente cualquier necesidad gráfica de una aplicación. Podemos destacar la manipulación de imágenes, gráficos vectoriales, animaciones, trabajo con texto o gráficos en 3D.

En este capítulo se introducen alguna de las características más significativas de la API gráfica de Android. Nos centraremos en el estudio de las clases utilizadas para el desarrollo de gráficos en 2D. En el capítulo 2 hemos descrito cómo se utilizan las vistas como elemento constructivo para el diseño de la interfaz de usuario. Disponíamos de una amplia paleta de vistas. No obstante, en muchas ocasiones va a ser interesante diseñar nuestras propias vistas. En este capítulo veremos cómo hacerlo.

Trataremos de aplicar lo aprendido en un ejemplo concreto, la representación de gráficos en Asteroides. Se utilizarán dos técnicas alternativas: los gráficos en mapa de bits y en formato vectorial. Al final del capítulo se describen las herramientas disponibles en Android para realizar animaciones. En concreto se describirán las animaciones Tween y las animaciones de propiedades. Por supuesto, resultaría imposible abarcar todas sus funciones para gráficos, por lo que se recomienda al lector que consulte la documentación de Android para obtener una descripción pormenorizada.



Objetivos:

- Enumerar las distintas API gráficas para 2D y 3D disponibles en Android.
- Describir cómo se utilizan las principales clases para gráficos en 2D (`Canvas`, `Paint` y `Path`).
- Introducir la clase `Drawable` y utilizar muchos de sus descendientes (`BitmapDrawable`, `GradientDrawable`, etc.).

- Estudiar cómo crear nuevas vistas y utilizarlas en distintas aplicaciones.
- Aplicar gran parte de lo aprendido en un ejemplo concreto: Asteroides.
- Describir las herramientas de Android para crear animaciones.

4.1. Clases para gráficos en Android

Antes de empezar a trabajar con gráficos, conviene familiarizarse con las clases que nos van a permitir crear y manipular gráficos en Android. A continuación se introducen algunas de estas clases:



Vídeo[tutorial]: *API para gráficos en Android*



Preguntas de repaso: *API para gráficos en Android*

4.1.1. Canvas

La clase `Canvas` representa una superficie donde podemos dibujar. Dispone de una serie de métodos que nos permiten representar líneas, círculos, texto, etc. Para dibujar en un `Canvas` necesitaremos un pincel (`Paint`), donde definiremos el color, el grosor de trazo, la transparencia, etc. También podremos definir una matriz de 3 × 3 (`Matrix`) que nos permitirá transformar coordenadas aplicando una translación, escala o rotación. Otra opción consiste en definir un área conocida como *Clip*, de forma que los métodos de dibujo afecten solo a esta área.

Veamos a continuación algunos métodos de la clase `Canvas`. No se trata de una lista exhaustiva, y muchos de estos métodos pueden trabajar con otros parámetros; por lo tanto, se recomienda consultar la documentación del SDK para una información más detallada.

Para dibujar figuras geométricas:

```
drawCircle(float cx, float cy, float radio, Paint pincel)
drawOval(RectF ovalo, Paint pincel)
drawRect(RectF rect, Paint pincel)
drawPoint(float x, float y, Paint pincel)
drawPoints(float[] pts, Paint pincel)
```

Para dibujar líneas y arcos:

```
drawLine(float iniX, float iniY, float finX, float finY,
                            Paint pincel)
drawLines(float[] puntos, Paint pincel)
```



```
drawArc(RectF ovalo, float iniAnglulo, float anglulo,
                          boolean usarCentro, Paint pincel)
drawPath(Path trazo, Paint pincel)
```

Para dibujar texto:

```
drawText(String texto, float x, float y, Paint pincel)
drawTextOnPath(String texto, Path trazo, float desplazamHor,
                   float desplazamVert, Paint pincel)
drawPosText(String texto, float[] posicion, Paint pincel)
      //Cada carácter se dibuja en la posición indicada
```

Para rellenar todo el `Canvas` (a no ser que se haya definido un *Clip*):

```
drawColor(int color)
drawARGB(int alfa, int rojo, int verde, int azul)
drawPaint(Paint pincel)
```

Para dibujar imágenes:

```
drawBitmap(Bitmap bitmap, Matrix matriz, Paint pincel)
```

Si definimos un *Clip*, solo se dibujará en el área indicada:

```
boolean clipRect(RectF rectangulo)
boolean clipRegion(Region region)
boolean clipPath(Path trazo)
```

Definir una matriz de transformación (`Matrix`) nos permitirá transformar coordenadas aplicando una translación, escala o rotación:

```
setMatrix(Matrix matriz)
Matrix getMatrix()
concat(Matrix matriz)
translate(float despazX, float despazY)
scale(float escalaX, float escalaY)
rotate(float grados, float centroX, float centroY)
skew(float despazX, float despazY)
```

Para averiguar el tamaño del `Canvas`:

```
int getHeight()
int getWidth()
```



Vídeo[tutorial]: *La clase Canvas en Android*



Ejercicio: *Creación de una vista personalizada*

A continuación, se muestra un ejemplo donde se crea una vista dibujada por código por medio de un `Canvas`.

1. Crea un nuevo proyecto con los siguientes datos:

```
Phone and Tablet / Empty Activity
Name: Ejemplo Graficos
Package name: org.example.ejemplograficos
Language: Java
Minimum API level: API 19 Android 4.4 (KitKat)
```

2. Reemplaza el código de la actividad por:

```
public class MainActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
         setContentView(new EjemploView(this));
    }

    public class EjemploView extends View {
         public EjemploView (Context context) {
            super(context);
         }
        @Override
         protected void onDraw(Canvas canvas) {
            //Dibujar aquí
         }
    }
}
```

Comienza con la creación de una `Activity`, pero en este caso, el objeto `View` que asociamos a la actividad mediante el método `setContentView()` no está definido mediante XML. Por el contrario, se crea mediante código a partir de la clase `EjemploView`.

La clase `EjemploView` extiende `View`, modificando solo el método `onDraw()` responsable de dibujar la vista. A lo largo del capítulo iremos viendo ejemplos de código que pueden ser escritos en este método.

3. Si ejecutas la aplicación, no se observará nada, ya que en el método `onDraw()` está vacío. Aprenderemos a dibujar en este `Canvas` utilizando el objeto `Paint` descrito en el siguiente apartado.



Nota sobre Java/Kotlin: *Siempre que en un ejemplo aparezca un error indicándote que no se puede resolver un tipo, pulsa **Alt-Intro** para añadir los* import.



Preguntas de repaso: *La clase Canvas en Android*

4.1.2. Paint

Como acabamos de ver, la mayoría de los métodos de la clase `Canvas` utilizan un parámetro de tipo `Paint`. Esta clase nos permite definir el color, estilo o grosor del trazado de un gráfico vectorial.



Vídeo[tutorial]: *La clase Paint en Android*



Ejercicio: *Uso de la clase Paint*

1. Escribe dentro de `onDraw` del ejercicio anterior el código siguiente:

```
Paint pincel = new Paint();
pincel.setColor(Color.BLUE);
pincel.setStrokeWidth(8);
pincel.setStyle(Style.STROKE);
canvas.drawCircle(150, 150, 100, pincel);
```



Nota sobre Java/Kotlin: *Si pulsas **Alt-Intro** para añadir los* `import`. *Puede que el sistema te advertirá de que la clase* `Style` *está definida en dos paquetes:*

Choose type to import   Page 1 of 1

android.graphics.Paint.Style
android.graphics.PathDashPathEffect.Style

Te preguntará cuál de los dos quieres importar. No suele resultar complicado resolver este problema si analizas el contexto en el que estás trabajando. En nuestro caso estamos utilizando la clase `Paint`, *por lo que la primera opción es la adecuada. Si alguna vez te equivocas en esta selección, lo normal es que aparezcan errores en el código. En ese caso, deshaces y seleccionas la otra opción.*

2. Ejecuta la aplicación para ver el resultado.
3. Aprovechando la opción de autocompletar y prueba otros valores para `Color` y `Style`.
4. Prueba otros métodos de dibujo, como `drawLine()` o `drawPoint()`.

Definición de colores

Android representa los colores utilizando enteros de 32 bits. Estos bits son divididos en 4 campos de 8 bits: alfa, rojo, verde y azul (ARGB, usando las iniciales en inglés). Al estar formado cada componente por 8 bits, podrá tomar 256 valores diferentes.

Los componentes rojo, verde y azul se utilizan para definir el color, mientras que el componente alfa define el grado de transparencia con respecto al fondo. Un valor de 255 significa un color opaco, y a medida que vayamos reduciendo el valor, el dibujo se irá haciendo transparente.

Para definir un color, tenemos las siguientes opciones:

```
int color;
color = Color.BLUE;                   //Azul opaco
color = Color.argb(127, 0, 255, 0);  //Verde transparente
color = 0x7F00FF00;          //Verde transparente
color = getResources().getColor(R.color.color_Circulo);
```

Para conseguir una adecuada separación entre programación y diseño, se recomienda utilizar la última opción. Es decir, no definir los colores directamente en código, si no utilizar el fichero de recursos `res/values/colors.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="color_circulo">#7fffff00</color>
</resources>
```

Como puedes observar, los colores han de definirse en el fichero `colors.xml` mediante sus componentes ARGB en hexadecimal.



Ejercicio: *Definición de colores*

1. Modifica el ejercicio anterior, añadiendo al final de `OnDraw` el código siguiente:

```
pincel.setColor(Color.argb(127,255,0,0));
canvas.drawCircle(150, 250, 100, pincel);
```

2. Observa como el color rojo seleccionado se mezcla con el color de fondo. Prueba otros valores de alfa.
3. Reemplaza la primera línea que acabas de introducir por:

```
pincel.setColor(0x7FFF0000);
```

4. Observa como el resultado es idéntico.
5. Define en `res/values/colors.xml` un nuevo color utilizando el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="color_circulo">#7fffff00</color>
</resources>
```


6. Modifica el ejemplo anterior para que se utilice este color definido en los recursos:

```
pincel.setColor(getResources().getColor(R.color.color_circulo));
```



Preguntas de repaso: *La clase Paint en Android*

4.1.3. Path

La clase `Path` permite definir un trazado a partir de segmentos de línea y curvas. Una vez definido, puede dibujarse con `canvas.drawPath(Path, Paint)`. Un `Path` también puede utilizarse para dibujar un texto sobre el trazado marcado.

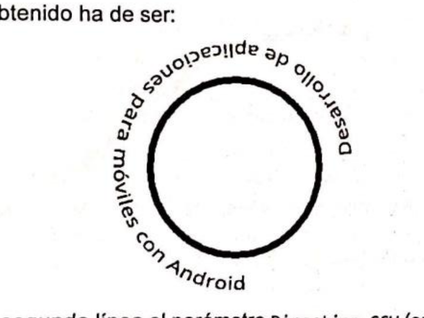


Ejercicio: *Uso de la clase Path*

1. Reemplaza dentro de `onDraw` del ejercicio anterior el código siguiente:

```
Path trazo = new Path();
trazo.addCircle(150, 150, 100, Direction.CCW);
canvas.drawColor(Color.WHITE);
Paint pincel = new Paint();
pincel.setColor(Color.BLUE);
pincel.setStrokeWidth(8);
pincel.setStyle(Style.STROKE);
canvas.drawPath(trazo, pincel);
pincel.setStrokeWidth(1);
pincel.setStyle(Style.FILL);
pincel.setTextSize(20);
pincel.setTypeface(Typeface.SANS_SERIF);
canvas.drawTextOnPath("Desarrollo de aplicaciones para
           móviles con Android", trazo, 10, 40, pincel);
```

2. El resultado obtenido ha de ser:



3. Modifica en la segunda línea el parámetro `Direction.CCW` (en sentido contrario a las agujas del reloj) por `Direction.CW` (en el sentido de las agujas del reloj). Observa el resultado.

4. Modifica los parámetros de `canvas.drawTextOnPath()` hasta que comprendas su significado.
5. ¿Podrías dibujar el texto en el sentido de las agujas del reloj por fuera del círculo?

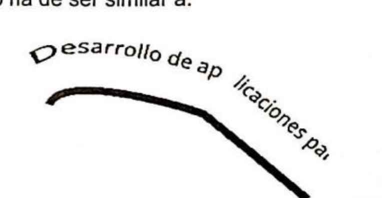


Ejercicio: *Un ejemplo de Path más complejo*

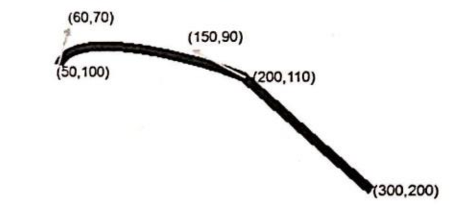
1. Reemplaza las dos primeras líneas del ejemplo anterior por:

```
Path trazo = new Path();
trazo.moveTo(50, 100);
trazo.cubicTo(60,70, 150,90, 200,110);
trazo.lineTo(300,200);
```

2. El resultado obtenido ha de ser similar a:



El trazo comienza desplazándose a las coordenadas (50,100). Luego introduce una curva cúbica o Bézier hasta la coordenada (200,110). Una curva Bézier introduce dos puntos de control: el primero (60,70) permite controlar cómo arranca la dirección del comienzo de la curva y el segundo (150,90), la dirección del final de la curva. Funciona de la siguiente manera: si trazas una recta desde el comienzo de la curva (50,100) hasta el primer punto de control (60,70), la curva se trazará tangencialmente a esta recta. Finalmente, se añade una línea desde las coordenadas (200,110) hasta (300,200).



3. ¿Por qué aparece una separación entre la «p» y la «l»? ¿Qué dos parámetros del siguiente gráfico tendríamos que modificar para que esta separación no estuviera? *(Solución: (150,90) = > (150,65).)*



Preguntas de repaso: *La clase Path en Android*

4.1.4. Drawable

La clase `Drawable` es una abstracción que representa «algo que se puede dibujar». Esta clase se extiende para definir una gran variedad de objetos gráficos más específicos. Muchos de ellos pueden definirse como recursos usando ficheros XML. Entre ellos tenemos los siguientes:

BitmapDrawable: Imagen basada en un fichero gráfico (PNG o JPG). Etiqueta XML `<bitmap>`.

VectorDrawable: Igual que `ShapeDrawable`, permite crear gráficos de forma vectorial. A diferencia de `ShapeDrawable`, podemos definir los gráficos en XML usando un formato basado en SVG (Scalable Vector Graphics). Este `Drawable` está disponible desde Android v5.0 (API 21), no obstante, se ha incluido en la librería de compatibilidad v7. Por lo tanto, podremos utilizarlo desde el API 7.

LayerDrawable: Contiene un *array* de `Drawable` que se visualizan según el orden del *array*. El índice mayor del *array* es el que se representa encima. Cada `Drawable` puede situarse en una posición determinada. Etiqueta XML `<layer-list>`.

StateListDrawable: Contiene un conjunto de `Drawable`, de manera que podemos indicar dinámicamente cuál de ellos es visible. Etiqueta XML `<selector>`.

GradientDrawable: Degradado de color que se puede ser usado en botones o fondos. Etiqueta XML `<gradient>`.

TransitionDrawable: Una extensión de `LayerDrawable` que permite un efecto de fundido entre la primera y la segunda capa. Para iniciar la transición hay que llamar a `startTransition(int tiempo)`. Para visualizar la primera capa hay que llamar a `resetTransition()`. Etiqueta XML `<transition>`.

ShapeDrawable: Permite realizar un gráfico a partir de primitivas vectoriales, como formas básicas (círculos, cuadrados...) o trazados (`Path`). Etiqueta XML `<shape>`. Crear un ShapeDrawable desde XML está muy limitado. Las opciones de dibujo se limitan a círculos o cuadrados. Para realizar un gráfico más complejo hemos de usar la clase `Path`, opción solo posible desde código.

AnimationDrawable: Permite crear animaciones *frame* a *frame* a partir de una serie de objetos `Drawable`. Etiqueta XML `<animation-list>`.

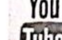
También puede ser interesante que uses la clase `Drawable` o uno de sus descendientes como base para crear tus propias clases gráficas.

Además de ser dibujada, la clase `Drawable` proporciona una serie de mecanismos genéricos que permiten indicar cómo ha de ser dibujada. No todo `Drawable` ha de implementar todos los mecanismos. Veamos los más importantes:

- `setBounds(x1,y1,x2,y2)` permite indicar el rectángulo donde ha de ser dibujado. Todo `Drawable` debe respetar el tamaño solicitado por el cliente, es decir, ha de permitir el escalado. Podemos consultar el tamaño preferido de un `Drawable` mediante los métodos `getIntrinsicHeight()` y `getIntrinsicWidth()`.

- `getPadding(Rect)` proporciona información sobre los **márgenes** recomendados para representar el contenido. Por ejemplo, un `Drawable` **que** intente ser un marco para un botón debe devolver los márgenes **correctos** para localizar las etiquetas, u otros contenidos, en el interior del botón.
- `setState(int[])` permite indicar al `Drawable` en qué estado ha de ser dibujado; por ejemplo, «con foco», «seleccionado», etc. Algunos `Drawable` cambiarán su representación según este estado.
- `setLevel(int)` algunos `Drawable` tienen un nivel asociado. Con este método podemos cambiarlo. Por ejemplo, un nivel puede interpretarse como una batería de niveles o un nivel de progreso. Algunos `Drawable` modificarán la imagen basándose en el nivel.

Un *drawable* puede realizar animaciones al ser llamado desde la interfaz `Drawable.Callback`. Tras implementar esta interfaz, hay que registrar un objeto de la clase creada llamando a `setCallback(Drawable.Callback)`.



Vídeo[tutorial]: *La clase Drawable en Android*



Preguntas de repaso: *La clase Drawable en Android*

Existen varias alternativas para crear una instancia de `Drawable`. Puedes crearla a partir de un fichero de imagen almacenado en los recursos del proyecto, también puedes crearla a partir del diseño basado en XML o puedes crearla a partir de código.

Veamos con más detalle algunas de las subclases de `Drawable`.

BitmapDrawable

La forma más sencilla de añadir gráficos a tu aplicación es incluirlos en la carpeta `res/drawable` del proyecto. El SDK de Android soporta los formatos PNG, JPG y GIF. El formato aconsejado es PNG, aunque si el tipo de gráficos así lo recomienda, también puedes utilizar JPG. El formato GIF está desaconsejado.

Cada gráfico de esta carpeta se asocia a un *id* de recurso. Por ejemplo, para el fichero `mi_imagen.png` se creará el *id* `mi_imagen`. Este *id* te permitirá hacer referencia al gráfico desde el código o desde XML.



Ejercicio: *Dibujar un BitmapDrawable de los recursos*

1. Busca en Internet un fichero gráfico en codificación *png* o *jpg* (los formatos gráficos usados por defecto en Android). Renombra el fichero para que se llame *mi_imagen.png* o *mi_imagen.jpg*.

2. En el proyecto del ejercicio anterior, arrastra el fichero a *res/drawable*.
3. Declara la variable `miImagen` en la clase `EjemploView` del ejercicio anterior:

```
private Drawable miImagen;
```

4. Escribe las siguientes tres líneas dentro del constructor de esta clase:

```
miImagen = AppCompatResources.getDrawable(context, R.drawable.mi_imagen);
miImagen.setBounds(30,30,200,200);
```

5. Escribe la siguiente línea en el método `onDraw`:

```
miImagen.draw(canvas);
```

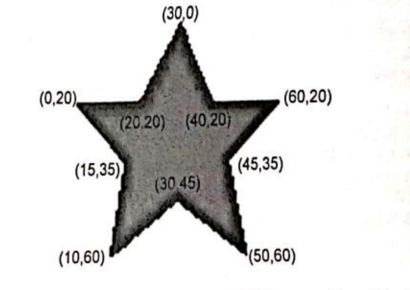
VectorDrawable

A partir de Android Lollipop (v5.0, API 21) se incorpora la posibilidad de definir drawables de forma vectorial utilizando un formato basado en SVG (Scalable Vector Graphics). También se dispone de una librería de compatibilidad que nos permite usar gráficos vectoriales en versiones anteriores. La gran ventaja de los gráficos vectoriales es que pueden ser reescalados sin perder definición. Solo necesitas un pequeño fichero para definir líneas y curvas, luego podrás representarlo en el tamaño deseado. Resulta mucho más sencillo que diseñar diferentes imágenes en mapa de bits para diferentes densidades.

Para definir un `VectorDrawable` con XML crea un nuevo fichero en *res/drawable*. Por ejemplo:

```
<vector xmlns:android="http://schemas.android.com/apk/res/android"
    android:width="60dp"
    android:height="60dp"
    android:viewportHeight="60"
    android:viewportWidth="60">
    <path
        android:fillColor="@color/colorPrimary"
        android:strokeColor="#000000"
        android:strokeWidth="2"
        android:pathData="M0 20 L20 20 L30 0 L40 20 L60 20 L45 35 L50 60
L30 45 L10 60 L15 35 Z" />
</vector>
```

Primero se define el ancho y alto con que queremos que aparezca el drawable y luego el ancho y alto con el que vamos a trabajar para definirlo. Con la etiqueta `<path>` definimos su trazado, comenzamos definiendo el color de relleno, color de trazo y grosor de trazo. Finalmente se indica los comandos que permiten dibujar la estrella mostrada a continuación. `M0 20` mueve el puntero de dibujo hasta estas coordenadas. `L20 20` dibuja una línea hasta estas coordenadas, ... Finalmente, `Z` cierra el trazado actual.



Para más detalles sobre estos comandos SVG consultar http://www.w3.org/TR/SVG11/paths.html#PathData. `VectorDrawable` no soporta algunas características de SVG, como los degradados. Consulta la documentación para ver todas las posibilidades tanto en Java como en XML[22].



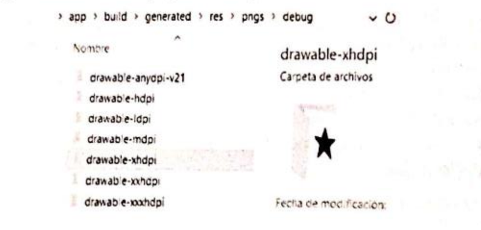
Ejercicio: *Dibujar un gráfico vectorial*

Android Studio permite usar gráficos vectoriales incluso en versiones anteriores a la 5.0. Para resolver el problema de compatibilidad con versiones anteriores, Android Studio detecta que estamos usando una versión mínima inferior a la 5.0; va a convertir los gráficos vectoriales a png. Veamos en este ejercicio como hacerlo.

1. Abre el proyecto del ejercicio anterior. Pulsa con el botón derecho sobre *res/drawable* y selecciona *New/Drawable resource file*. Introduce como nombre *estrella.xml*. Reemplaza su contenido por el que se acaba de mostrar.
2. En la clase `EjemploView` del ejercicio anterior, haz el siguiente cambio:

`miImagen=AppCompatResources.getDrawable(context,R.drawable.~~mi_imagen~~.estrella);`

3. Ejecuta la aplicación y verifica el resultado.
4. En el explorador del proyecto pulsa con el botón derecho sobre *app* y selecciona *Show in explorer* (o *Show in Dolfin*). Dentro de la carpeta del proyecto accede a *app / build / generated / res / pngs / debug*.



[22] http://developer.android.com/intl/es/reference/android/graphics/drawable/VectorDrawable.html

Observa como en la carpeta `drawable-anydpi-v21` se guarda el fichero XML. Este recurso será usado en versiones API 21 o superior, para cualquier densidad gráfica. En el resto de carpetas se ha generado un .png de forma automática. Este recurso será usado en versiones anteriores a la 21, según la densidad del dispositivo.

NOTA: En caso de que el proyecto especifique una versión mínima de API igual o superior a la 21, estos pngs ya no serán generados.

5. El gráfico ha sido diseñado para ser dibujado con 20 dp. Y esta información es la que utiliza el sistema para generar los pngs. Veamos qué pasa si lo dibujamos a un tamaño mucho mayor. Para ello cambia la siguiente línea:

`miImagen.setBounds(30,30,~~200~~1000,~~200~~1000);`

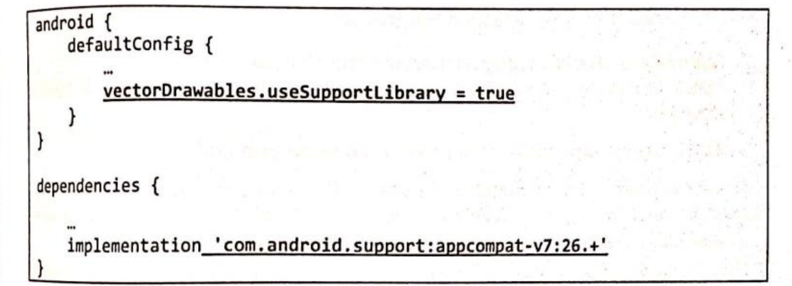
5. Ejecuta la aplicación en un dispositivo con nivel de API 21 o superior. Observa como el reescalado es perfecto.
6. Ejecuta la aplicación en un dispositivo con nivel de API inferior a 21. Observa como el reescalado es deficiente, apareciendo el efecto de pixelado.



Ejercicio: *Usar la librería de compatibilidad para VectorDrawable*

Como acabamos de ver, Android Studio convierte automáticamente los gráficos vectoriales a png para que puedan ser utilizados en versiones anteriores a la 5.0. Esta solución tiene sus limitaciones, no podemos aumentar el tamaño, cambiar los atributos del gráfico dinámicamente (como uno de sus colores) y, además, ocupan más memoria. En este ejercicio, usaremos otra alternativa: usar una librería de compatibilidad.

1. Para usar gráficos vectoriales en versiones anteriores al API 21, añade al fichero *build.gradle(Module:app)*:



```
android {
    defaultConfig {
        …
        vectorDrawables.useSupportLibrary = true
    }
}

dependencies {
    …
    implementation 'com.android.support:appcompat-v7:26.+'
}
```

La primera línea indica a Android Studio que vamos a utilizar la librería de compatibilidad para Vector Drawable. La segunda línea es posible que ya esté añadida en el proyecto.

2. Borra la carpeta *app / build / generated*. Selecciona la opción *Build / Make Project*. Observa como ahora ya no se ha generado la carpeta *app / build /*

generated / res / pngs. Ya no es necesario que se conviertan los ficheros vectoriales en png. Ahora, se podrá hacer en tiempo de ejecución por software.

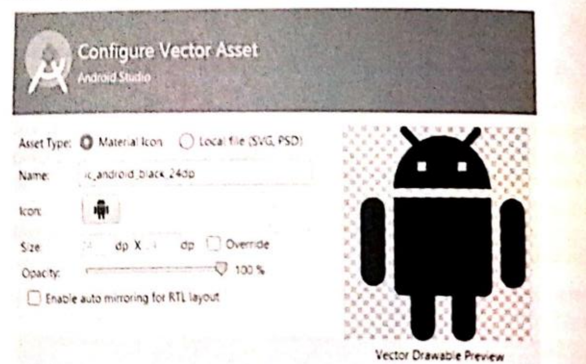
3. Ejecuta la aplicación en un dispositivo con nivel de API inferior a 21. Observa como el reescalado es perfecto.



Ejercicio: *Agregar gráficos vectoriales con Vector Asset*

En el primer ejercicio de esta sección vimos cómo crear `VectorDrawable` mediante su código XML. En la práctica esto no se suele así así. Cuando necesitamos un gráfico podemos buscarlo en una colección, o si tenemos que generarlo nosotros usaremos un editor de gráficos vectoriales. Para ayudarnos en este trabajo, Android Studio incorpora la herramienta Vectos Asset. Veamos cómo se usa en este ejercicio.

1. En Android Studio selecciona *File / New / Vector Asset*.



2. Seleccionando *Material Icon*, y pulsa en el icono. Puedes acceder a una colección de iconos. Selecciona uno muy sencillo, por ejemplo, *ic_arrow_drop_up*. Pulsa *Next* y luego *Finish*.
3. Modifica el ejercicio anterior para que se visualice este gráfico.
4. Abre el fichero XML que acabas de añadir. Observa con que poca información se ha definido el gráfico. Cambia el color y algún valor del path. Verifica que el resultado es el esperado.
5. Busca en Internet y fichero con extensión SVG (Scalable Vector Graphics) o PSD (Adobe Photoshop).
6. Abre de nuevo Vectos Asset y selecciona *Local file (SVG, PSD)*. En el campo *Name* indica un nombre de recurso y en *Path* el fichero que acabas de descargar. Si lo deseas también puedes cambiar el tamaño que tendrá por defecto la imagen y el grado de opacidad.

7. El formato XML usado en VectorDrawable no soporta todas las características disponibles en los formatos SVG y PSD. En el cuadro *Errors*, que encontrarás en la parte inferior, se mostrará un listado con los problemas en la conversión.
8. Observa como convierte el fichero al formato utilizado en Android.

La gran ventaja de usar gráficos vectoriales es que ocupan muy poco espacio, podemos cambiar fácilmente los colores y no tenemos que preocuparnos de preparar diferentes recursos para diferentes resoluciones. Como inconveniente los gráficos vectoriales pueden costar más recursos para ser generados, especialmente si son complejos y tienen curvas.



Preguntas de repaso: *BitmapDrawable y VectorDrawable*

GradientDrawable

También podemos definir en XML otro tipo de `Drawables`, como `GradientDrawable`. Por ejemplo, el siguiente fichero define un degradado desde el color blanco (`FFFFFF`) a azul (`0000FF`):

```
<shape xmlns:android="http://schemas.android.com/apk/res/android">
  <gradient
    android:startColor="#FFFFFF"
    android:endColor="#0000FF"
    android:angle="270" />
</shape>
```

Este tipo de objetos gráficos se utiliza con frecuencia como fondo de botones o de pantalla. El parámetro `angle` indica la dirección del degradado. Solo se permiten ángulos múltiplos de 45 grados.



Ejercicio: *Definir un GradientDrawable*

1. Abre el proyecto Asteroides.
2. Crea el fichero *res/drawable/degradado.xml* con el código anterior.
3. Podrías introducir la siguiente línea en el constructor de una vista, para conseguir que este *drawable* sea utilizado como fondo:

```
setBackgroundResource(R.drawable.degradado);
```

4. Resulta más conveniente definir el fondo de una vista en su *layout* en XML en lugar de hacerlo por código. Comenta la línea introducida en el punto anterior e introduce el siguiente atributo en el *layout main.xml*:

```
android:background="@drawable/degradado"
```

TransitionDrawable

Un `TransitionDrawable` es una extensión de `LayerDrawable` que permite un efecto de fundido entre la primera y la segunda capa. Para iniciar la transición, hay que llamar a `startTransition(int tiempo)`. Para volver a visualizar la primera capa, hay que llamar a `resetTransition()`.



Ejercicio: *Definir un TransitionDrawable*

1. Crea un nuevo proyecto con nombre *Transition* y tipo *Empty Activity*.
2. Crea el siguiente recurso en *res/drawable/transicion.xml* con el código:

```
<?xml version="1.0" encoding="utf-8"?>
<transition xmlns:android=
      "http://schemas.android.com/apk/res/android">
  <item android:drawable="@drawable/asteroide1"/>
  <item android:drawable="@drawable/asteroide3"/>
</transition>
```

3. Copia los ficheros *asteroide1.png* y *asteroide3.png* a la carpeta *res/drawable*. Puedes encontrar los gráficos en el siguiente link:

   http://www.androidcurso.com/images/dcomg/ficheros/Graficos.zip

4. Reemplaza en la actividad el método `onCreate` por el siguiente código:

```
@Override public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   ImageView image = new ImageView(this);
   setContentView(image);
   TransitionDrawable transition = AppCompatResources.getDrawable(this,
                                                  R.drawable.transicion);
   image.setImageDrawable(transition);
   transition.startTransition(2000);
}
```

5. Si todo funciona correctamente, verás como la llamada a `transition.startTransition(2000)` provoca que la primera imagen se transforme a lo largo de dos segundos en la segunda imagen.

ShapeDrawable

Cuando quieras crear un gráfico dinámicamente mediante primitivas vectoriales, una buena opción puede ser utilizar `ShapeDrawable`. Esta clase permite dibujar gráficos a partir de formas básicas. Un `ShapeDrawable` es una extensión de `Drawable`; por lo tanto, puedes utilizar todo lo que permite `Drawable`.



Ejercicio: *Definir un ShapeDrawable*

Veamos un ejemplo de cómo utilizar un objeto `ShapeDrawable` para crear una vista a medida.

1. Abre el proyecto EjemploGraficos.
2. En la clase `EjemploView` declara la siguiente variable:

```
private ShapeDrawable miImagen;
```

3. Añade las siguientes tres líneas dentro del constructor de esta clase:

```
miImagen = new ShapeDrawable(new OvalShape());
miImagen.getPaint().setColor(0xff0000ff);
miImagen.setBounds(10, 10, 310, 60);
```

4. En el constructor, un objeto `ShapeDrawable` es definido como un óvalo. Se le asigna un color y se definen sus fronteras.
5. Escribe la siguiente línea en el método `onDraw`:

```
miImagen.draw(canvas);
```

6. Ejecuta la aplicación y observa el resultado.

AnimationDrawable

Android nos proporciona varios mecanismos para crear animaciones. Una ventaja que destacar es que estas animaciones pueden ser creadas en ficheros XML. En este apartado veremos una de las animaciones más sencillas, las creadas a partir de una serie de fotogramas. Para ello utilizaremos la clase `AnimationDrawable`.



Ejercicio: *Uso de AnimationDrawable*

1. Crea un nuevo proyecto con nombre *Animación* y tipo *Empty Activity*.
2. Crea el siguiente recurso *res/drawable/animacion.xml*:

```
<animation-list
  xmlns:android= "http://schemas.android.com/apk/res/android"
  android:oneshot= "false">
    <item android:drawable="@drawable/misil1"
          android:duration="200" />
    <item android:drawable="@drawable/misil2"
          android:duration="200" />
    <item android:drawable="@drawable/misil3"
          android:duration="200" />
</animation-list>
```

3. Copia los ficheros *misil1.png*, *misil2.png* y *misil3.png* a la carpeta *res/drawable*. Puedes encontrar los gráficos en el siguiente link:

   http://www.androidcurso.com/images/dcomg/ficheros/Graficos.zip

4. Reemplaza el código de la actividad por:

```
public class MainActivity extends Activity {
   AnimationDrawable animacion;

   @Override public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      animacion = AppCompatResources.getDrawable(this,
R.drawable.animacion);
      ImageView vista = new ImageView(this);
      vista.setBackgroundColor(Color.WHITE);
      vista.setImageDrawable(animacion);
      vista.setOnClickListener(new OnClickListener() {
         public void onClick(View view) {
            animacion.start();
         }
      });
      setContentView(vista);
   }
}
```

5. En este ejemplo se comienza declarando un objeto `animacion` de la clase `AnimationDrawable`. Se inicializa utilizando el fichero XML incluido en los recursos. Se crea una nueva vista de la clase `ImageView` para ser representada por la actividad y se pone como imagen de esta vista `animacion`. Finalmente, se crea un escuchador de evento `onClick` para que cuando se pulse sobre la vista se ponga en marcha la animación.

   NOTA: *A partir de Android API 19 (4.4) este tipo de animaciones se inician automáticamente, mientras que en las anteriores hace falta iniciarlas manualmente.*



Preguntas de repaso: *Otros tipos de Drawable*

4.2. Creación de una vista en un fichero independiente

Como hemos visto en los ejemplos anteriores, para poder dibujar en Android hemos tenido que crear una nueva clase `EjemploView`, descendiente de View. Esta clase se creaba dentro de `MainActivity`, por lo que solo podía utilizarse desde esta clase. Va a resultar mucho más interesante crear esta clase en un fichero independiente. De esta forma podremos utilizarla desde cualquier parte de nuestro proyecto, o desde otros proyectos. Incluso estará disponible en la paleta de vistas del editor visual. El siguiente ejercicio te permite verificar este concepto.



Ejercicio: *Creación de una nueva vista independiente*

En este ejercicio vamos a poner la clase `EjemploView` en un fichero independiente para que pueda utilizarse desde cualquier parte.

1. Abre el proyecto EjemploGraficos.
2. En la clase `MainActivity` selecciona todo el texto correspondiente a la definición de la clase `EjemploView` y córtalo. Se almacenará en el portapapeles.
3. Pulsa con el botón derecho sobre *org.example.ejemplografico* y selecciona la opción *New/Class...* Introduce como nombre de la clase `EjemploView`.
4. Pega el texto que has puesto en el portapapeles.
5. Ejecutar la aplicación y verifica que el resultado es idéntico al obtenido en el apartado anterior.

En este apartado aprovechamos para introducir otros aspectos que deberás tener en cuenta para crear nuevas vistas. Cuando quieres crear una nueva vista, tendrás que extender la clase `View`, escribir un constructor y, como mínimo, sobrescribir los métodos `onSizeChanged()` y `onDraw()`. Es decir, tendrás que seguir el siguiente esquema:

```
public class MiVista extends View {

   public MiVista(Context context, AttributeSet attrs) {
      super(context, attrs);

      //Inicializa la vista
      //OjO: Aún no se conocen sus dimensiones
   }

   @Override protected void onSizeChanged(int ancho, int alto,
         int ancho_anterior, int alto_anterior){
      //Te informan del ancho y la altura
   }

   @Override protected void onDraw(Canvas canvas) {
      //Dibuja aquí la vista
   }
}
```

Observa como el constructor utilizado tiene dos parámetros: el primero, de tipo `Context`, te permitirá acceder al contexto de aplicación (por ejemplo, para utilizar recursos de esta aplicación). El segundo, de tipo `AttributeSet`, te permitirá acceder a los atributos de esta vista, cuando sea creada desde XML. El constructor es el lugar adecuado para crear todos los componentes de tu vista; pero, cuidado: en este punto todavía no se sabe qué dimensiones tendrá.

Android realiza un proceso de varias pasadas para determinar la anchura y altura de cada vista dentro de un *layout*. Cuando finalmente ha establecido las

dimensiones de una vista, llamará a su método `onSizeChanged()`. Nos indica como parámetros la anchura y altura asignadas. En caso de tratarse de un reajuste de tamaños (por ejemplo, una de las vistas del *layout* desaparece y el resto tienen que ocupar su espacio), se nos pasará la anchura y altura anterior. Si es la primera vez que se llama al método, estos parámetros valdrán 0.

El último método que siempre tendrás que rescribir es `onDraw()`. Es aquí donde tendrás que dibujar la vista.



Ejercicio: *Una vista que pueda diseñarse desde XML*

Vamos a modificar la vista anterior para que se pueda utilizar usando un diseño en XML:

1. Modifica el código de la clase `EjemploView` para que coincida con el siguiente (en negrita se resaltan las diferencias):

```
public class EjemploView extends View {
   private ShapeDrawable miImagen;

   public EjemploView(Context context, AttributeSet attrs) {
      super(context, attrs);
      miImagen = new ShapeDrawable(new OvalShape());
      miImagen.getPaint().setColor(0xff0000ff);
   }

   @Override protected void onSizeChanged(int ancho, int alto,
         int ancho_anterior, int alto_anterior){
      miImagen.setBounds(0, 0, ancho, alto);
   }
   @Override protected void onDraw(Canvas canvas) {
      miImagen.draw(canvas);
   }
}
```

La primera diferencia es la utilización de un constructor con el parámetro `AttributeSet`. Este parámetro es imprescindible si quieres definir la vista en XML. También se ha añadido el método `onSizeChanged()`, para que el óvalo se dibuje siempre ajustado al tamaño de la vista.

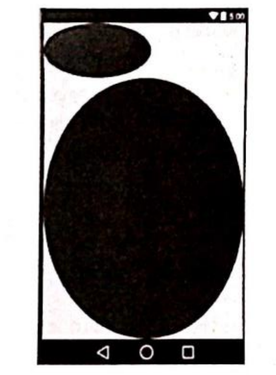
2. Abre el fichero *activity_main.xml* y reemplaza su contenido por el siguiente:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical">
   <org.example.ejemplograficos.EjemploView
         android:id="@+id/ejemploView1"
         android:layout_width="400px"
         android:layout_height="200px" />
   <org.example.ejemplograficos.EjemploView
         android:id="@+id/ejemploView2"
         android:layout_width="match_parent"
         android:layout_height="match_parent" />
</LinearLayout>
```

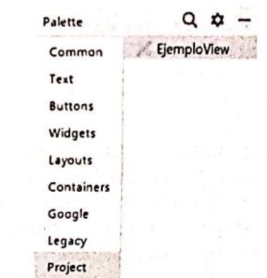
NOTA: No utilices el valor `"wrap_content"`. *Nuestra vista no ha sido programada para soportar este tipo de valor.*

3. En la clase `MainActivity` reemplaza `setContentView(new EjemploView(this))` por `setContentView(R.layout.activity_main)`.

4. Ejecuta la aplicación. El resultado ha de ser similar a:

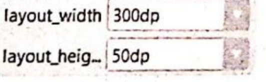


5. Esta nueva vista también puede ser insertada en un *layout* usando el editor visual. Pulsa en la lengüeta *Design* para pasar a este modo de edición. En la paleta de vistas, busca la sección *Project*.



6. Selecciona *EjemploView* y arrástralo entre los dos óvalos.

7. Modifica en la ventana *Attributes* los siguientes valores:



4.3. Creando la actividad principal de Asteroides

Una vez que conocemos los rudimentos que nos permiten utilizar gráficos en Android, vamos a aplicarlos a nuestro ejemplo.



Ejercicio: *Creando la actividad principal de Asteroides*

Lo primero que necesitamos es crear una actividad que controle la pantalla del juego propiamente dicho. A esta actividad la llamaremos `Juego`.

1. Abre el proyecto Asteroides.
2. Crea la clase `Juego` con el siguiente código.

```
public class Juego extends Activity {
   @Override public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.juego);
   }
}
```

3. Necesitaremos un *layout* para la pantalla del juego. Crea el fichero `res/layout/juego.xml` con el siguiente contenido:

```
<LinearLayout
     xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent" >
   <org.example.asteroides.VistaJuego
      android:id="@+id/VistaJuego"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:focusable="true"
      android:background="@drawable/fondo" />
</LinearLayout>
```

Como puedes observar, cuando diseñamos un *layout* en XML, no estamos limitados a escoger los elementos que tenemos en la paleta de vistas: vamos a utilizar vistas creadas por nosotros. En este ejemplo utilizaremos la vista `org.example.asteroides.VistaJuego`, que se describirá más adelante. Va a ser esta vista la que lleve el peso de la aplicación.

4. Observa que se ha indicado el parámetro `android:background` asociado al recurso `@drawable/fondo`. Por lo tanto, tendremos que poner el recurso *fondo.jpg* en la carpeta correspondiente. Copia también los gráficos correspondientes a los asteroides, la nave y el misil en la carpeta `res/drawable`. Los puedes encontrar en el siguiente link. Estos gráficos se utilizarán en los siguientes apartados.

http://www.androidcurso.com/images/dcomg/ficheros/Graficos.zip

5. De momento no podremos ejecutar la aplicación hasta haber definido la clase `VistaJuego`.

4.3.1. La clase Gráfico

El juego que estamos desarrollando va a desplazar varios tipos de gráficos por pantalla: asteroides, nave, misiles, etc. Dado que el comportamiento de todos estos elementos es muy similar, con el fin de reutilizar el código y mejorar su comprensión, vamos a crear una clase que represente un gráfico a desplazar por pantalla. A esta nueva clase la llamaremos `Grafico` y presentará las siguientes características. El elemento a dibujar será representado en un objeto `Drawable`. Como hemos visto, esta clase presenta una gran versatilidad, lo que nos permitirá trabajar con gráficos en *bitmap* (`BitmapDrawable`), vectoriales (`ShapeDrawable`), gráficos con diferentes representaciones (`StateListDrawable`), gráficos animados (`AnimationDrawable`), etc. Además, un `Grafico` dispondrá de posición, velocidad de desplazamiento, dirección y velocidad de rotación. Como característica especial, un gráfico que salga por uno de los extremos de la pantalla reaparecerá por el extremo opuesto, tal y como ocurría en el juego original de Asteroides.



Ejercicio: *La clase Gráfico*

1. Crea la clase `Grafico` en el proyecto Asteroides con el siguiente código:

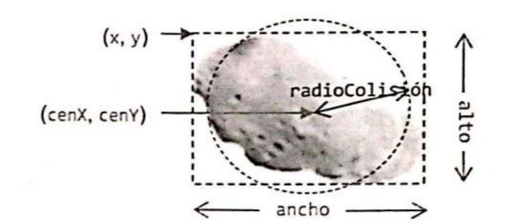
```
class Grafico {
  private Drawable drawable;   //Imagen que dibujaremos
  private int cenX, cenY;      //Posición del centro del gráfico
  private int ancho, alto;     //Dimensiones de la imagen
  private double incX, incY;   //Velocidad desplazamiento
  private double angulo, rotacion;//Ángulo y velocidad rotación
  private int radioColision;   //Para determinar colisión
  private int xAnterior, yAnterior; // Posición anterior
  private int radioInval;       // Radio usado en invalidate()
  private View view;             // Usada en view.invalidate()

  public Grafico(View view, Drawable drawable){
    this.view = view;
    this.drawable = drawable;
    ancho = drawable.getIntrinsicWidth();
    alto = drawable.getIntrinsicHeight();
    radioColision = (alto+ancho)/4;
    radioInval = (int) Math.hypot(ancho/2, alto/2);
  }

  public void dibujaGrafico(Canvas canvas){
    int x = cenX - ancho/2;
    int y = cenY - alto/2;
    drawable.setBounds(x, y, x+ancho, y+alto);
    canvas.save();
    canvas.rotate((float)angulo, cenX, cenY);
    drawable.draw(canvas);
    canvas.restore();
```

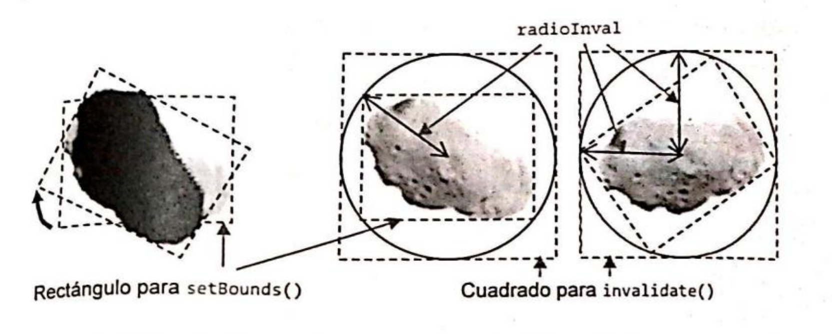
```
        xAnterior = cenX;
        yAnterior = cenY;
    }
    public void incrementaPos(double factor){
        cenX += incX * factor;
        cenY += incY * factor;
        angulo += rotacion * factor;
        // Si salimos de la pantalla, corregimos posición
        if(cenX<0)              cenX=view.getWidth();
        if(cenX>view.getWidth())  cenX=0;
        if(cenY<0)              cenY=view.getHeight();
        if(cenY>view.getHeight()) cenY=0;
        view.postInvalidate (cenX-radioInval, cenY-radioInval,
                    cenX+radioInval, cenY+radioInval);
        view.postInvalidate(xAnterior-radioInval, yAnterior-radioInval,
                    xAnterior+radioInval, yAnterior+radioInval);
    }
    public double distancia(Grafico g) {
        return Math.hypot(cenX-g.cenX, cenY-g.cenY);
    }
    public boolean verificaColision(Grafico g) {
        return (distancia(g) < (radioColision + g.radioColision));
    }
}
```

Cada objeto de la clase `Grafico` se caracteriza por situarse en unas coordenadas centrales (`cenX,cenY`). Por lo tanto, su esquina superior izquierda estará en las coordenadas (`x, y`) = (`cenX - ancho/2, cenY - alto/2`). Aunque los gráficos pueden ser alargados (si `alto` diferente de `ancho`), a efectos de las colisiones, vamos a considerar que son círculos. El radio de este círculo se podría calcular como `ancho/2` o `alto/2`, según que tomemos el radio mayor o el radio menor. Lo que hacemos es tomar una media de estos dos posibles radios: `(ancho/2+alto/2)/2 = (ancho+alto)/4`. El método `verificaColision()` comprueba si hemos colisionado con otro gráfico. Para ello se comprueba si la distancia al otro `Grafico` es menor a la suma de los dos radios de colisión.



El método `dibujaGrafico()` se encarga de dibujar el `Drawable` del `Grafico` en un `Canvas`. Comienza indicando los límites donde se situará el `Drawable`, utilizando `setBounds()`. Luego, guarda la matriz de transformación del `Canvas`. A continuación aplica una transformación de rotación según lo indicado en la variable `angulo` utilizando como eje de rotación (`cenX, cenY`). Se dibuja el `Drawable` en el `Canvas` y se recupera la matriz de transformación, para que la rotación introducida no se aplique en futuras operaciones con este `Canvas`.

Para finalizar, en caso de que el gráfico se haya movido, hacemos dos llamadas al método `invalidate()` de la vista donde estamos dibujando el `Grafico`. Con este método informamos a la vista de que tiene que ser redibujada. Para mejorar la eficiencia indicaremos solo el rectángulo que hemos modificado. De esta forma, la vista no tendrá que redibujarse en su totalidad. En un primer momento podríamos pensar que el rectángulo de invalidación coincide con el utilizado en `setBounds()`. Pero hay que recordar que hemos realizado una rotación sobre el `Drawable` y, como puede verse en la ilustración de la izquierda, posiblemente el `Drawable` se salga de este rectángulo. Para resolver este problema vamos a aumentar el área de invalidación a un cuadrado con la mitad de su lado igual a la mitad de la diagonal del gráfico. Este valor es precalculado en la variable `radioInval`.



Hay que tener en cuenta que hemos desplazado el `Drawable` desde una posición anterior. Por lo tanto, también es necesario indicar a la vista que redibuje el área donde estaba antes el `Grafico`. Con este fin vamos a utilizar las varibles `xAnterior` y `yAnterior`.

Otro método interesante es `incrementaPos()`, que se utiliza para modificar la posición y el ángulo del gráfico según la velocidad de translación (`incX, incY`) y la velocidad de rotación (`rotacion`). Este método tiene el parámetro `factor`, que permite ajustar esta velocidad. Con un valor igual a 1, tendremos una velocidad normal; si vale 2, el gráfico se mueve al doble de velocidad. En el juego original de Asteroides, si un gráfico salía por un lado de la pantalla, aparecía de nuevo por el lado opuesto. Este comportamiento se implementa en las últimas líneas del método.

2. Al principio de la clase hemos definido varios campos con el modificador `private`. Vamos a necesitar acceder a estos campos desde fuera de la clase, por lo que resulta necesario declarar los métodos *get* y *set* correspondientes. Vamos a realizar esta tarea de forma automática utilizando una herramienta de Android Studio. Sitúa el cursor al final de la clase (justo antes de la última llave) y pulsa con el botón derecho. Selecciona en el menú desplegable *Generate... > Getters and Setters...* En la ventana de diálogo marca todos los campos y pulsa *OK*. Android Studio hará el trabajo por nosotros.



Nota sobre Java: *En el tutorial Java Esencial > Encapsulamiento y visibilidad* puedes aprender más sobre los métodos get y set. Lo encontrarás en la web www.androidcurso.com.

4.3.2. La clase VistaJuego

Pasemos a describir la creación de `VistaJuego`, que como hemos indicado es la responsable de la ejecución del juego. En una primera versión solo se representarán los asteroides de forma estática.



Ejercicio: *La clase VistaJuego*

1. Crea una nueva clase `VistaJuego` en el proyecto Asteroides y copia el siguiente código:

```
public class VistaJuego extends View {
   // //// ASTEROIDES //////
   private List<Grafico> asteroides; // Lista con los Asteroides
   private int numAsteroides = 5; // Número inicial de asteroides
   private int numFragmentos = 3; // Fragmentos en que se divide

   public VistaJuego(Context context, AttributeSet attrs) {
      super(context, attrs);
      Drawable drawableNave, drawableAsteroide, drawableMisil;
      drawableAsteroide = AppCompatResources.getDrawable(context,
                                          R.drawable.asteroide1);
      asteroides = new ArrayList<Grafico>();
      for (int i = 0; i < numAsteroides; i++) {
         Grafico asteroide = new Grafico(this, drawableAsteroide);
         asteroide.setIncY(Math.random() * 4 - 2);
         asteroide.setIncX(Math.random() * 4 - 2);
         asteroide.setAngulo((int) (Math.random() * 360));
         asteroide.setRotacion((int) (Math.random() * 8 - 4));
         asteroides.add(asteroide);
      }
   }

   @Override protected void onSizeChanged(int ancho, int alto,
                              int ancho_anter, int alto_anter) {
      super.onSizeChanged(ancho, alto, ancho_anter, alto_anter);
      // Una vez que conocemos nuestro ancho y alto.
      for (Grafico asteroide: asteroides) {
         asteroide.setCenX((int) (Math.random()*ancho));
         asteroide.setCenY((int) (Math.random()*alto));
      }
   }

   @Override protected void onDraw(Canvas canvas) {
      super.onDraw(canvas);
```



```
      for (Grafico asteroide: asteroides) {
         asteroide.dibujaGrafico(canvas);
      }
   }
}
```

Como ves, se han declarado tres métodos. En el constructor creamos los asteroides e inicializamos su velocidad, dirección y velocidad de rotación. Sin embargo, resulta imposible iniciar su posición, dado que no conocemos la altura y anchura de la pantalla. Esta información se conocerá cuando se llame a `onSizeChanged()`. Observa como en esta función los asteroides están situados de forma aleatoria. El último método, `onDraw()`, es el más importante de la clase `View`, dado que es el responsable de dibujar la vista.

2. Hemos creado una vista personalizada. No tendría demasiado sentido, pero podrá utilizarse en cualquier otra aplicación que desarrolles. Visualiza el *layout* `juego.xml` y observa como la nueva vista se integra perfectamente en el entorno de desarrollo.



3. Registra la actividad `Juego` en *AndroidManifest.xml.*
4. Asocia al atributo `onClick` del botón *Jugar* del *layout activity_main.xml* el método `lanzarJuego()`. Crea el método `lanzarJuego()` dentro de la clase `MainActivity`, de forma similar al método `lanzarAcercaDe()`, pero esta vez arrancando la actividad `Juego`.
5. Ejecuta la aplicación. Has de ver cinco asteroides repartidos al azar por la pantalla.

4.3.3. Introduciendo la nave en VistaJuego

El siguiente paso consiste en dibujar la nave que controlará el usuario para destruir los asteroides.



Ejercicio: *Introduciendo la nave en VistaJuego*

1. Declara las siguientes variables al comienzo de la clase `VistaJuego`:

```
// //// NAVE //////
private Grafico nave; // Gráfico de la nave
```

```
private int giroNave; // Incremento de dirección
private double aceleracionNave; // aumento de velocidad
private static final int MAX_VELOCIDAD_NAVE = 20;
// Incremento estándar de giro y aceleración
private static final int PASO_GIRO_NAVE = 5;
private static final float PASO_ACELERACION_NAVE = 0.5f;
```

Algunas de estas variables se utilizarán en el siguiente capítulo.

2. En el constructor de la clase instancia la variable `drawableNave` de forma similar a como se ha hecho en `drawableAsteroide`.
3. Inicializa también en el constructor la variable `nave` de la siguiente forma:

```
nave = new Grafico(this, drawableNave);
```

4. En el método `onSizeChange()` posiciona la nave justo en el centro de la vista.
5. En el método `onDraw()` dibuja la nave en el `Canvas`.
6. Ejecuta la aplicación. La nave ha de aparecer centrada:



7. Si cuando situamos los asteroides, alguno coincide con la posición de la nave, el jugador no tendrá ninguna opción de sobrevivir. Sería más interesante asegurarnos de que al posicionar los asteroides estos se encuentran a suficiente distancia de la nave, y en caso contrario, tratar de obtener una nueva posición del asteroide. Para conseguirlo puedes utilizar el siguiente código en sustitución de las dos líneas `asteroide.setCenX(…)` y `asteroide.setCenY(…)`dentro de la clase `VistaJuego`:

```
do {
   asteroide.setCenX((int) (Math.random()*ancho));
   asteroide.setCenY((int) (Math.random()*alto));
} while(asteroide.distancia(nave) < (ancho+alto)/5);
```

***NOTA:** Ten cuidado donde añades este código.*

Ejercicio: *Evitando que VistaJuego cambie su representación con el dispositivo en horizontal y en vertical*

1. Ejecuta la aplicación y cambia de orientación la pantalla del dispositivo. En el emulador se consigue pulsando la tecla *Ctrl-F11*.
2. Observa cómo, cada vez, se reinicializa la vista, regenerando los asteroides. Esto nos impediría jugar de forma adecuada. Para solucionarlo, edita *AndroidManifest.xml*. En la lengüeta `Application` selecciona la actividad `Juego`. En los parámetros de la derecha selecciona en *Screen orientation: landscape*. Observa como en el código XML se ha añadido el siguiente atributo:

```
android:screenOrientation="landscape"
```

3. Ejecuta de nuevo la aplicación. Observa como la actividad Juego será siempre representada en modo horizontal, de forma independiente de la posición del teléfono.

   ***NOTA:** En algunos emuladores, cuando presiones Ctrl-F11 la orientación seguirá cambiando. Se trata de un error de simulación, al no soportar esta configuración. En ese caso, cambia de emulador o pruébalo en un dispositivo real.*

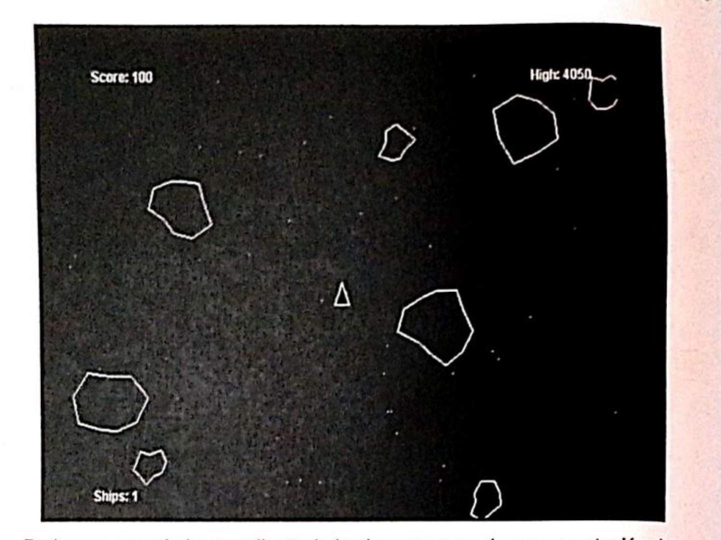
4. Abre de nuevo las propiedades de la actividad `Juego`. En *Theme* selecciona el valor `@android:style/Theme.NoTitleBar.Fullscreen`. Este tema visualizará la vista ocupando toda la pantalla, sin la barra de notificaciones ni el nombre de la aplicación.
5. Si en *Theme* pulsas el botón *Browse…* y seleccionas el botón circular *System Resources*, puedes ver una lista de estilos definidos en el sistema.

   ***NOTA:** En algunas instalaciones esta lista puede que no te funcione.*

6. Ejecuta la aplicación en un terminal real y verifica el resultado.

4.4. Representación de gráficos vectoriales en Asteroides

La versión original del juego Asteroides fue escrita para ordenadores con escasa potencia gráfica. Tal y como puedes ver a continuación, nuestra nave se representaba con un simple triángulo y los asteroides, como polígonos irregulares.



Dado que, cuando hemos diseñado la clase `Grafico`, la representación de este la hemos delegado en un objeto `Drawable`, va a resultar muy fácil cambiar los gráficos de la aplicación para que tengan una apariencia más retro. Simplemente, utilizando la subclase de `Drawable`, `ShapeDrawable`, en lugar de `BitmapDrawable`, para cambiar la forma de dibujar los gráficos.



Ejercicio: *Representación vectorial de los asteroides*

1. Abre la clase `VistaJuego`.
2. En el constructor reemplaza la línea:

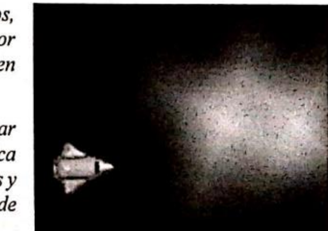
```
drawableAsteroide =
     AppCompatResources.getDrawable(context, R.drawable. asteroide1);
```

por el siguiente código:

```
SharedPreferences pref = PreferenceManager.
                      getDefaultSharedPreferences(getContext());
if (pref.getString("graficos", "1").equals("0")) {
   Path pathAsteroide = new Path();
   pathAsteroide.moveTo((float) 0.3, (float) 0.0);
   pathAsteroide.lineTo((float) 0.6, (float) 0.0);
   pathAsteroide.lineTo((float) 0.6, (float) 0.3);
   pathAsteroide.lineTo((float) 0.8, (float) 0.2);
   pathAsteroide.lineTo((float) 1.0, (float) 0.4);
   pathAsteroide.lineTo((float) 0.8, (float) 0.6);
   pathAsteroide.lineTo((float) 0.9, (float) 0.9);
   pathAsteroide.lineTo((float) 0.8, (float) 1.0);
   pathAsteroide.lineTo((float) 0.4, (float) 1.0);
   pathAsteroide.lineTo((float) 0.0, (float) 0.6);
   pathAsteroide.lineTo((float) 0.0, (float) 0.2);
   pathAsteroide.lineTo((float) 0.3, (float) 0.0);
   ShapeDrawable dAsteroide = new ShapeDrawable(
                          new PathShape(pathAsteroide, 1, 1));
   dAsteroide.getPaint().setColor(Color.WHITE);
   dAsteroide.getPaint().setStyle(Paint.Style.STROKE);
   dAsteroide.setIntrinsicWidth(50);
   dAsteroide.setIntrinsicHeight(50);
   drawableAsteroide = dAsteroide;
   setBackgroundColor(Color.BLACK);
} else {
   drawableAsteroide =
        AppCompatResources.getDrawable(context, R.drawable.asteroide1);
}
```


Lo primero que hace este código es consultar en las preferencias para ver si el usuario ha escogido gráficos vectoriales. En caso negativo se realizará la misma inicialización de `drawableAsteroide` que teníamos antes. En caso afirmativo comenzamos creando la variable `pathAsteroide` de la clase `Path`. En este objeto se introducen todas las órdenes de dibujo necesarias para dibujar un asteroide. Luego se crea la variable `dAsteroide` de la clase `ShapeDrawable` para crea un *drawable* a partir del *path*. Los últimos dos parámetros (...,`1`,`1`) significan el valor de escala aplicado al eje *x* y al eje *y*. Luego se indica el color y el estilo del pincel e indicamos la altura y anchura por defecto del *drawable*. Finalmente asignamos el objeto creado a `drawableAsteroide`.

3. Ejecuta la aplicación y selecciona el tipo de gráficos adecuado en las preferencias.



NOTA: En algunos dispositivos físicos, cuando se activa la aceleración gráfica por hardware, los asteroides pueden representarse de forma algo extraña:

Para evitar este problema puede resultar interesante desactivar la aceleración grafica cuando trabajemos con gráficos vectoriales y activarla en caso contrario. Par ello, añade dentro del `if` *del código anterior:*

```
setLayerType(View.LAYER_TYPE_SOFTWARE, null);
```

y dentro de `else`*:*

```
setLayerType(View.LAYER_TYPE_HARDWARE, null);
```

Otra posible solución consiste en desactivar en AndroidManifest.xml la aceleración gráfica por hardware. Para ello en la etiqueta `<activity>` *correspondiente a la actividad* `Juego` *añade el atributo:*

```
android:hardwareAccelerated="false"
```



Desafío: *Representación de los asteroides con VectorDrawable*

A partir de la versión 5 de Android, es posible definir un gráfico vectorial en un recurso XML. Esta forma de trabajar resulta mucho más interesante que la propuesta en el ejercicio "Representación vectorial de los asteroides". Dado que permite separar del código aspectos relacionados con el diseño de la aplicación. Por ejemplo, el diseñador gráfico de nuestra empresa podrá modificar directamente la forma de un asteroide, sin tener que acceder al código.

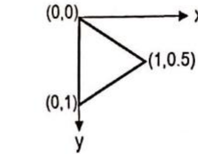
1. Añade a las preferencias un nuevo tipo de gráficos. Además de vectorial y bitmap, se ha de poder escoger la opción VectorDrawable.
2. Crea tres recursos de tipo `VectorDrawable` que representen tres asteroides de tamaños distintos. Para esto puedes tienes varias opciones.
   a) Puedes copiar el XML de la estrella y modificar el path usando los puntos definidos en "Representación vectorial de los asteroides".
   b) Puedes usar un editor vectorial como Adobe Photoshop o SVG-Edit (https://github.com/SVG-Edit/svgedit, pulsar en *Try SVG-edit here*.) y crear tus propios asteroides.
   c) Puedes buscar en Internet algún fichero SVG con forma de asteroide.
3. Modifica el código para que al seleccionar como tipo de gráficos VectorDrawable, se utilicen estos recursos.



Práctica: *Representación vectorial de la nave*

Como habrás comprobado en el ejercicio anterior, la nave se representa siempre utilizando un fichero *png*. En esta práctica has de intentar que también pueda representarse vectorialmente.

1. Crea un nuevo objeto de la clase `Path` para representar la nave dentro de la sección `if` introducida en el ejercicio anterior. Como puedes ver en la ilustración siguiente, ha de ser un simple triángulo. Como el ángulo de rotación inicial es cero, la nave ha de mirar a la derecha.



2. Crea un nuevo `ShapeDrawable` a partir del `Path` anterior. Unas dimensiones adecuadas para la nave pueden ser 20 de ancho y 15 de alto.
3. Inicializa la variable `drawableNave` de forma adecuada.



Desafío: *Representación de la nave con VectorDrawable*

Realiza el mismo trabajo propuesto en el desafío "Representación de los asteroides con VectorDrawable", pero ahora para representar la nave.

4.5. Animaciones

El entorno de programación Android incorpora tres mecanismos para crear animaciones en nuestras aplicaciones:

- La clase `AnimationDrawable`: Permite crear *drawables* que reproducen una animación fotograma a fotograma. Se ha descrito su uso en el apartado sobre `Drawables`.
- Animaciones de vistas (también conocidas como animaciones Tween): Permiten crear efectos de translación, rotación, zum y transparencia en cualquiera vista de nuestra aplicación. Se describen a continuación.
- Animaciones de propiedades: Nuevo mecanismo incorporado en Android 3.0. Permiten animar cualquier propiedad de cualquier objeto, sea una vista o no. Además, modifican el objeto en sí, no solamente cambia su representación en pantalla como ocurre en una animación Tween. Se describen en *El gran libro de Android Avanzado*.

Los siguientes apartados describen con más detalle las animaciones de vistas, y se hace una introducción de las animaciones de propiedades:

4.5.1. Animaciones de vistas

Una animación de vista o Tween puede realizar series de transformaciones simples (posición, tamaño, rotación y transparencia) en el contenido de un objeto View. Por ejemplo, si tienes un `TextView` puedes moverlo, rotarlo, aumentarlo, disminuirlo o cambiarle la transparencia al texto.

La secuencia de órdenes que define la "animación Tween" puede estar escrita mediante XML o código, pero es recomendable el fichero XML, al ser más legible, reutilizable e intercambiable.

Las instrucciones de la animación definen las transformaciones que quieres que ocurran, cuándo ocurrirán y cuánto tiempo tardarán en completarse. Las transformaciones pueden ser secuenciales o simultáneas. Cada tipo de transformación tiene unos parámetros específicos, y también existen unos parámetros comunes a todas las transformaciones, como el tiempo de inicio y la duración.

El fichero XML que define la animación debe pertenecer al directorio *res/anim/* en tu proyecto Android. El archivo debe tener un único elemento raíz, que debe ser uno de los siguientes: `<translate>`, `<rotate>`, `<scale>`, `<alpha>` o elemento `<set>`, que puede contener grupos de estos elementos (además de otro `<set>`). Por defecto,

todas las instrucciones de animación ocurren a partir del instante inicial. Si quieres que una animación comience más tarde, debes especificar el atributo startOffset.



Ejercicio: *Creación de una animación Tween para animar una vista*

1. Crea un nuevo proyecto con nombre *AnimacionTween* y tipo *Empty Activity*.
2. Crea el fichero *res/anim/animacion.xml* y pega el siguiente código:

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <scale      android:duration="2000"
                android:fromXScale="2.0"
                android:fromYScale="2.0"
                android:toXScale="1.0"
                android:toYScale="1.0" />
    <rotate     android:startOffset="2000"
                android:duration="2000"
                android:fromDegrees="0"
                android:toDegrees="360"
                android:pivotX="50%"
                android:pivotY="50%"/>
    <translate  android:startOffset="4000"
                android:duration="2000"
                android:fromXDelta="0"
                android:fromYDelta="0"
                android:toXDelta="50"
                android:toYDelta="100" />
    <alpha      android:startOffset="4000"
                android:duration="2000"
                android:fromAlpha="1"
                android:toAlpha="0" />
</set>
```

3. Abre el fichero *res/layout/ activity_main.xml* y añade el siguiente atributo a la vista de tipo TextView:

```
        android:id="@+id/textView"
```

4. Abre la actividad del proyecto y añade las líneas marcadas en negrita al método onCreate().

```
@Override public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout. activity_main);
    TextView texto = findViewById(R.id.textView);
    Animation animacion = AnimationUtils.loadAnimation(this,
                                          R.anim.animacion);
    texto.startAnimation(animacion);
}
```

5. Ejecuta la aplicación.

6. Como podrás ver, el TextView comienza haciéndose más pequeño (etiqueta <scale>), después rota sobre sí mismo (etiqueta <rotate>) y finalmente se desplaza (etiqueta <translate>) a la vez que se hace transparente (etiqueta <alpha>). Al finalizar la animación, vuelve a su posición y estado inicial, sin importar dónde ni cómo haya acabado.



Recursos adicionales: *Lista de etiquetas de las animaciones de vistas y sus atributos*

Los siguientes atributos son aplicables a todas las transformaciones:

startOffset – Instante inicial de la transformación en milisegundos.

duration – Duración de la transformación en milisegundos.

repeatCount – Número de repeticiones adicionales de la animación.

interpolator – En lugar de realizar una transformación lineal, se aplica algún tipo de interpolación. Algunos de los valores posibles son:

accelerate_decelerate_interpolator, accelerate_interpolator, anticipate_interpolator, anticipate_overshoot_interpolator, bounce_interpolator, cycle_interpolator, decelerate_interpolator, linear_interpolator, overshoot_interpolator

Lista de las transformaciones con sus atributos específicos:

<translate> – Desplaza la vista

fromXDelta, toXDelta – Valor inicial y final del desplazamiento en eje X.

fromYDelta, toYDelta – Valor inicial y final del desplazamiento en eje Y.

<rotate> – Rota la vista.

fromDegrees, toDegrees – Valor inicial y final en grados de la rotación en grados. Si quieres un giro completo en sentido antihorario, pon de 0 a 360, y si lo quieres en sentido horario, de 360 a 0 o de 0 a -360. Si quieres dos giros, pon de 0 a 720.

pivotX, pivotY – Punto sobre el que se realizará el giro. Este quedará fijo en la pantalla.

<scale> – Cambia el tamaño de la vista.

fromXScale, toXScale – Valor inicial y final para la escala del eje X (0.5 = 50 %, 1 = 100 %).

fromYScale, toYScale – Valor inicial y final para la escala del eje Y.

pivotX, pivotY – Punto sobre el que se realizará el zoom. Este quedará fijo en la pantalla.

<alpha> – Cambia la opacidad de la vista.

fromAlpha, toAlpha – Valor inicial y final de la opacidad.



Práctica: *Introduciendo animaciones en Asteroides*

En esta práctica has de conseguir que los diferentes elementos del *layout* inicial de Asteroides vayan apareciendo uno tras otro con diferentes efectos.

1. Abre el proyecto Asteroides.
2. Crea una nueva animación y llámala *giro_con_zoom.xml*. Ha de durar dos segundos y de forma simultánea ha de hacer un zum de escala 3 a 1 y un giro de dos vueltas (720º). El punto de anclaje de la rotación y el zum han de ser el centro de la vista.
3. Selecciona el *layout activity_main.xml* y pon un *id* al `TextView` correspondiente al título.
4. En la actividad inicial de Asteroides, crea un objeto correspondiente a este `TextView` y aplícale la animación anterior.
5. Crea una nueva animación y llámala *aparecer.xml*. Ha de comenzar a los dos segundos, durar un segundo y modificar el valor de *alpha* de 0 hasta 1.
6. Aplica esta animación al primer botón.
7. Crea una nueva animación y llámala *desplazamiento_derecha.xml*. Ha de comenzar a los tres segundos, durar un segundo y modificar el valor de *desplazamiento x* de 400 hasta 0. Prueba también algún tipo de interpolación
8. Aplica esta animación al segundo botón.
9. Si dispones de tiempo, crea dos nuevas animaciones a tu gusto y aplícalas al tercer y cuarto botón.
10. Aplica la animación *giro_con_zoom.xml* al botón *Acerca de* cuando sea pulsado. Observa como al lanzar la nueva actividad `AcercaDeActivity`, la actividad principal continúa ejecutándose.

4.5.2. Animaciones de propiedades

A partir de la versión 3.0 de Android (nivel de API 11) se ha incorporado un nuevo tipo de animaciones. A diferencia de las animaciones Tween, que solo son aplicables a vistas, una animación de propiedades puede animar cualquier tipo de objeto. Además, no está restringida a las cuatro transformaciones antes vistas: podemos animar cualquier propiedad del objeto. Por ejemplo, podemos hacer una animación que cambie progresivamente el color de fondo de una vista.

Otra diferencia con respecto a las animaciones Tween es que estas solo modifican la forma en que la vista es representada, pero no sus propiedades. Por ejemplo, si aplicas una animación Tween para que un texto se desplace por la pantalla, se visualizará correctamente, pero al acabar la animación el texto estará en el lugar inicial, lo que te obligará a implementar tu propia lógica para manejar este cambio de posición. En una animación de propiedades estará cambiando el objeto en sí, no solamente cómo se representa.

Desventajas de las animaciones Tween:

- Solo podemos animar objetos de la clase `View`.
- Están limitadas a estas cuatro transformaciones: translación, rotación, escalado y transparencia. No pueden aplicarse a otros efectos, como cambiar el color de fondo.
- Solo modifican la forma en que la vista es representada, pero no sus propiedades en sí.

Desventajas de las animaciones de propiedades:

- Solo disponibles a partir de la versión 3.0.
- Requieren más tiempo en inicializarse y hay que escribir más código.

Para aprender más sobre este tipo de animaciones, puedes leer el siguiente apartado de *Android Developers*: **Property Animation**[23] o el primer capítulo de *El Gran Libro de Android Avanzado*.

[23] http://developer.android.com/guide/topics/graphics/prop-animation.html

CAPÍTULO 5.
Hilos de ejecución, pantalla táctil y sensores

La forma más habitual para interactuar con un ordenador es el teclado y el ratón. Por desgracia, estos dispositivos de entrada no existen, o están muy limitados, en un teléfono móvil. No obstante, los nuevos móviles permiten nuevas formas de interacción con el usuario, por lo que el diseño de nuestras aplicaciones ha de adaptarse a estas nuevas formas de interacción. A lo largo de este capítulo se estudiarán diferentes alternativas para recoger las acciones que los usuarios realizan sobre la aplicación.

Este capítulo comienza describiendo la importancia de los hilos de ejecución (*threads*) en Android. Se indica cómo y cuándo hay que crear nuevos hilos. Tras una visión general del manejo de eventos en Android, continuaremos con algunos dispositivos de entrada: el teclado, la pantalla táctil y los sensores. Estos tres mecanismos de interacción se aplicarán al manejo de nuestra nave en la aplicación Asteroides.



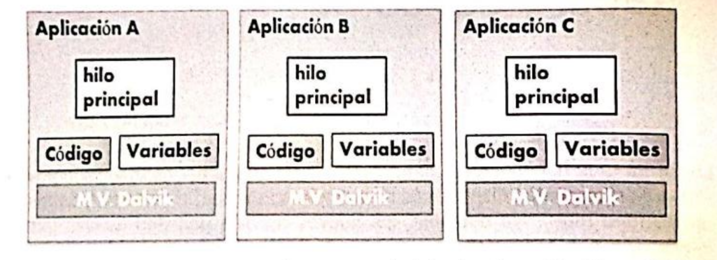
Objetivos:

- Describir el uso de hilos de ejecución (*threads*).
- Aprender a crear nuevos hilos usando las clases `Thread` y `AsyncTask`.
- Mostrar las distintas alternativas para manejar los eventos de usuario en Android.
- Describir cómo se manejan los eventos del teclado.
- Aprender a interaccionar con la pantalla táctil.
- Enumerar los sensores disponibles en muchos terminales Android y aprender a utilizarlos.

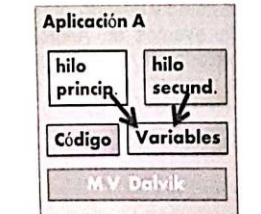
5.1. Uso de hilos de ejecución (*threads*)

5.1.1. Introducción a los procesos e hilos de ejecución

Cada vez que se lanza una nueva aplicación en Android, el sistema crea un nuevo proceso Linux para ella y la ejecuta en su propia máquina virtual Dalvik o ART (por supuesto, si está programada en Java; si lo estuviera en código nativo, no haría falta la máquina virtual). Trabajar en procesos diferentes nos garantiza que desde una aplicación no se pueda acceder a la memoria (código o variables) de otras aplicaciones. Como se verá en un próximo capítulo, esta característica se hereda directamente del sistema operativo Linux.



Los SO modernos incorporan el concepto de hilo de ejecución (*thread*). En un sistema multihilo, un proceso va a poder realizar varias tareas a la vez, cada una en un hilo diferente. Los diferentes hilos de un proceso lo comparten todo: variables, código, permisos, ficheros abiertos, etc.



Cuando trabajamos con varios hilos, estos pueden acceder a las variables de forma simultánea. Hay que tener cuidado de que un hilo no modifique el valor de una variable mientras otro hilo la está leyendo. Este problema se resuelve en Java definiendo secciones críticas mediante la palabra reservada `synchronized`. Trataremos este problema más adelante.

5.1.2. Hilos de ejecución en Android

Cuando se lanza una nueva aplicación, el sistema crea un nuevo hilo de ejecución (*thread*) para esta aplicación, conocido como hilo principal. Este hilo es muy importante, dado que se encarga de atender a los eventos de los distintos componentes. Es decir, este hilo ejecuta los métodos `onCreate()`, `onDraw()`,

`onKeyDown()`, etc. El hilo principal también es el responsable de atender a los eventos generados desde la interfaz de usuario. Por esta razón, al hilo principal también se lo conoce como hilo de la interfaz de usuario.

El sistema no crea un hilo independiente cada vez que se crea un nuevo componente. Es decir, todas las actividades y servicios de una aplicación son ejecutados por el hilo principal de la aplicación.

Cuando tu aplicación ha de realizar trabajo intensivo como respuesta a una interacción de usuario, hay que tener cuidado porque es posible que la aplicación no responda de forma adecuada. Por ejemplo, imagina que has de esperar a que finalice la realización de algún cálculo que lleve bastante tiempo, como por ejemplo verificar si un entero muy grande es primo o no; si lo haces en el hilo de ejecución principal, este quedará bloqueado a la espera de que termine el cálculo. Por lo tanto, no se podrá redibujar la vista (`onDraw()`) o atender a eventos del usuario (`onKeyDown()`). Desde el punto de vista del usuario, se tendrá la impresión de que la aplicación se ha colgado. Más todavía: si el hilo principal se bloquea más de 5 segundos, el sistema mostrará al usuario el cuadro de diálogo *"La aplicación no responde"* para que el usuario decida si quiere esperar o detener la aplicación.

La solución en estos casos es crear un nuevo hilo de ejecución, para que realice este trabajo intensivo. De esta forma no bloqueamos el hilo principal, que puede seguir atendiendo a los eventos de usuario. Es decir, cuando estés implementando un método que es ejecutado por el hilo principal (suelen empezar por `on...`), no realices nunca una acción que pueda bloquear este hilo, como cálculos largos o que requieran esperar mucho tiempo. En estos casos hay que crear un nuevo hilo de ejecución y encomendarle esa tarea. En el siguiente apartado describiremos cómo hacerlo.

Las herramientas de la interfaz de usuario de Android han sido diseñadas para ser ejecutadas desde un único hilo de ejecución, el hilo principal. Por lo tanto, no se permite manipular la interfaz de usuario desde otros hilos de ejecución.



Vídeo[**tutorial**]: *Hilos de ejecución en Android*



Ejercicio: *Una tarea que bloquea el hilo principal*

En muchas ocasiones hemos de realizar costosas operaciones o hemos de esperar a que concluyan lentas operaciones en la red. En ambos casos, hay que tener la precaución de no bloquear el hilo principal. De hacerlo, el resultado puede ser catastrófico, como se muestra en el siguiente ejercicio.

1. Crea un nuevo proyecto con nombre *Hilos* y tipo *Empty Activity*.
2. Reemplaza el código del *layout activity_main* por:

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >
  <LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
    <EditText
      android:id="@+id/entrada"
      android:layout_width="0dip"
      android:layout_height="wrap_content"
      android:layout_weight="1"
      android:inputType="numberDecimal"
      android:text="5" >
      <requestFocus />
    </EditText>
    <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:onClick="calcularOperacion"
      android:text="Calcular factorial" />
  </LinearLayout>
  <TextView
  android:id="@+id/salida"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:text=" "
  android:textAppearance="?android:attr/textAppearanceMedium"/>
</LinearLayout>
```

3. Reemplaza el código de MainActivity por el siguiente:

```
public class MainActivity extends AppCompatActivity {
  private EditText entrada;
  private TextView salida;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    entrada = findViewById(R.id.entrada);
    salida = findViewById(R.id.salida);
  }

  public void calcularOperacion(View view) {
    int n = Integer.parseInt(entrada.getText().toString());
    salida.append(n +"! = ");
    int res = factorial(n);
    salida.append(res +"\n");
  }

  public int factorial(int n) {
    int res=1;
```



```
    for (int i=1; i<=n; i++){
      res*=i;
      SystemClock.sleep(1000);
    }
    return res;
  }
}
```

El método `calcularOperacion()` se llamará cuando se pulse el botón. Comienza obteniendo el valor entero introducido en `entrada` y muestra la operación a realizar por el `TextView salida`. Luego, se llama al `factorial()` y se muestra el resultado.

El método `factorial()` calcula la operación matemática factorial de un entero *n* (*n*!). Se calcula multiplicando todos los enteros desde uno hasta *n*. Por ejemplo: factorial(5) = 5! = 1 · 2 · 3 · 4 · 5 = 120. En el código, esta operación se calcula con un bucle con la variable `i` tomando valores de 1 hasta `n`. Esta operación se va a computar de forma muy rápida y apenas bloquearemos el hilo principal unas milésimas de segundo. Para que aparezca el problema que estamos estudiando, vamos a simular que se realizan un gran número de operaciones en cada pasada del bucle. Para ello llamaremos a `SystemClock.sleep(1000)`, que bloqueará el hilo durante 1000 ms (1 segundo).

4. Ejecuta la aplicación y calcula el factorial de 5. El resultado ha de ser similar al siguiente:



Observa cómo, mientras se realiza la operación, el usuario no puede pulsar el botón ni modificar el `EditText`. El usuario tendrá la sensación de que la aplicación está bloqueada.

5. Calcula ahora el factorial de 10. Si mientras está calculando interaccionas con la interfaz de usuario, el sistema nos acabará mostrando el siguiente error:

Hilos no responde
× Cerrar aplicación
◯ Esperar

Mensajes del tipo "La aplicación no responde" son frecuentes en Android. Aparecen cuando el hilo principal se bloquea demasiado tiempo. En el siguiente apartado mostraremos como realizar esta operación de forma correcta.

5.1.3. Creación de nuevos hilos con la clase Thread

Como acabamos de ver, siempre que tengamos que ejecutar un método que requiera bastante tiempo de ejecución, no podremos ejecutarlo en el hilo principal. Dado que este hilo ha de estar siempre disponible para atender a los eventos generados por el usuario, nunca debe ser bloqueado. En este apartado aprenderemos a crear nuevos hilos usando la clase de Java `Thread`. El proceso es muy sencillo, no tendremos más que escribir una clase como la siguiente:

```
class MiThread extends Thread {
    @Override
    public void run() {
        …
    }
}
```

El método `run()` contiene el código que queremos que el hilo ejecute. Para crear un nuevo hilo y ejecutarlo escribiremos:

```
MiThread hilo = new MiThread();
hilo.start();
```

La llamada al método `start()` ocasionará que se ejecute el método `run()` del hilo. La llamada a `start()` es asíncrona, es decir, continuaremos ejecutando las instrucciones siguientes, sin esperar a que el método `run()` concluya. Como veremos más adelante, si esperamos algún resultado de este método, será imprescindible establecer algún mecanismo de sincronización para saber cuándo ha terminado.



Ejercicio: *Crear un nuevo hilo con la clase Thread*

En este ejercicio ejecutaremos el método `factorial()` en un hilo nuevo. Además, veremos algunas limitaciones de usar nuevos hilos, como la imposibilidad de acceder a la interfaz de usuario.

1. Abre el proyecto creado en el ejercicio anterior.
2. Dentro de `MainActivity` introduce el siguiente código:

```
class MiThread extends Thread {
   private int n, res;

   public MiThread(int n) {
      this.n = n;
   }

   @Override public void run() {
      res = factorial(n);
      salida.append(res + "\n");
   }
}
```

Siguiendo el esquema mostrado anteriormente, hemos creado un hilo que en su método `run()` llama a `factorial()` y muestra el resultado por pantalla. Para realizar la operación necesitamos el parámetro de entrada `n`. Este no puede incorporarse el método `run()`, dado que ha de ser sobrescrito (`@Override`) sin alteración alguna. Para resolverlo hemos añadido un constructor a la clase, donde se indica este parámetro.

3. Reemplaza el siguiente método en `MainActivity`:

```
public void calcularOperacion(View view) {
   int n = Integer.parseInt(entrada.getText().toString());
   salida.append(n + "! = ");
   MiThread thread = new MiThread(n);
   thread.start();
}
```

4. Ejecuta la aplicación. Tras pulsar el botón, el resultado ha de ser:

La aplicación Hilos ha dejado de funcionar

Volver a abrir la aplicación

5. Abre la vista *LogCat* y busca el siguiente error:

```
FATAL EXCEPTION: Thread-157
android.view.ViewRootImpl$CalledFromWrongThreadException: Only th ↵
e original thread that created a view hierarchy can touch its vie ↵
ws.
```

Este mensaje indica que solo desde el hilo principal se puede interactuar con las vistas de la interfaz de usuario. ***NOTA:** También está prohibido desde otros hilos usar la clase Toast.*

Una forma elegante de resolver este problema podría ser usar la clase `Callable`. Esta clase de Java nos permite llamar un método en un nuevo hilo, esperar a que este termine y recoger los resultados. No obstante, nuestro objetivo es mostrar las peculiaridades de Android con el manejo de hilos, así que vamos a resolver el problema de otra forma.

6. En el método `run()` reemplaza:

```
salida.append(res + "\n");
```

por:

```
runOnUiThread(new Runnable() {
   @Override public void run() {
      salida.append(res + "\n");
   }
});
```

De esta forma indicamos al sistema que ejecute parte de nuestro código en el hilo principal.

7. Ejecuta la aplicación y comprueba que el resultado es satisfactorio.



Preguntas de repaso: *Hilos de ejecución*

5.1.4. Introduciendo movimiento en Asteroides

Para que el juego cobre vida será necesario animar todos los gráficos introducidos. En el siguiente ejercicio veremos cómo hacerlo.



Ejercicio: *Introduciendo movimiento en Asteroides*

1. Comienza declarando las siguientes variables en la clase `VistaJuego`:

```
// //// THREAD Y TIEMPO //////
// Thread encargado de procesar el juego
private ThreadJuego thread = new ThreadJuego();
// Cada cuanto queremos procesar cambios (ms)
private static int PERIODO_PROCESO = 50;
// Cuando se realizó el último proceso
private long ultimoProceso = 0;
```

2. La animación del juego la llevará a cabo el método `actualizaFisica()`, que será ejecutado a intervalos regulares definidos por la constante `PERIODO_PROCESO`. Esta constante ha sido inicializada a 50 ms. En `ultimoProceso` se almacena el instante en que se llamó por última vez a `actualizaFisica()`.

3. Copia el siguiente método dentro de la clase `VistaJuego`:

```
protected void actualizaFisica() {
   long ahora = System.currentTimeMillis();
   if (ultimoProceso + PERIODO_PROCESO > ahora) {
      return;    // Salir si el período de proceso no se ha cumplido.
   }
   // Para una ejecución en tiempo real calculamos el factor de movimiento
   double factorMov = (ahora - ultimoProceso) / PERIODO_PROCESO;
   ultimoProceso = ahora; // Para la próxima vez
   // Actualizamos velocidad y dirección de la nave a partir de
   // giroNave y aceleracionNave (según la entrada del jugador)
   nave.setAngulo((int) (nave.getAngulo() + giroNave * factorMov));
   double nIncX = nave.getIncX() + aceleracionNave *
             Math.cos(Math.toRadians(nave.getAngulo())) * factorMov;
   double nIncY = nave.getIncY() + aceleracionNave *
            Math.sin(Math.toRadians(nave.getAngulo())) * factorMov;
   // Actualizamos si el módulo de la velocidad no excede el máximo
   if (Math.hypot(nIncX,nIncY) <= MAX_VELOCIDAD_NAVE){
      nave.setIncX(nIncX);
      nave.setIncY(nIncY);
   }
   nave.incrementaPos(factorMov); // Actualizamos posición
   for (Grafico asteroide : asteroides) {
      asteroide.incrementaPos(factorMov);
   }
}
```


Como veremos a continuación, se llamará a este método de forma continua para visualizar la animación. Como queremos desplazar los gráficos cada `PERIODO_PROCESO` milisegundos, verificamos si ya ha pasado este tiempo desde la última vez que se ejecutó (`ultimoProceso`).

Como también es posible que el sistema esté ocupado y no nos haya podido llamar hasta un tiempo superior a `PERIODO_PROCESO`, vamos a calcular el factor de movimiento en función del tiempo adicional que haya pasado. Si, por ejemplo, desde la última llamada ha pasado dos veces `PERIODO_PROCESO`, la variable `factorMov` ha de valer 2. Lo que significará que los gráficos han de desplazarse el doble que en circunstancias normales. De esta forma conseguiremos un desplazamiento continuo en tiempo real.

A continuación se actualizan las variables que controlan la dirección de la nave y la velocidad. Se consiguen por medio de las variables `giroNave` y `aceleracionNave`. En este capítulo modificaremos estas variables para que el jugador pueda pilotar la nave. A partir de estas variables se obtiene una nueva velocidad de la nave, descompuesta en sus componentes *x* e *y* (`nIncX` y `nIncY`). Si el módulo de estos componentes es mayor que la velocidad máxima permitida, no se actualizará la velocidad.

Finalmente se actualizan las posiciones de todos los gráficos (nave y asteroides) a partir de sus velocidades. Esto se consigue llamando al método `incrementaPos()` definido en la clase `Grafico`.

4. Ahora necesitamos que esta función sea llamada continuamente, para lo que utilizaremos un *thread*. Crea la siguiente clase dentro de la clase `VistaJuego`:

```
class ThreadJuego extends Thread {
   @Override
   public void run() {
      while (true) {
         actualizaFisica();
      }
   }
}
```

5. Introduce estas líneas al final del método `onSizeChanged()`:

```
ultimoProceso = System.currentTimeMillis();
thread.start();
```

Esto ocasionará que se llame al método `run()` del hilo de ejecución. Este método es un bucle infinito que continuamente llama al `actualizaFisica()`.

6. Ejecuta la aplicación y observa como el juego cobra vida. Como los valores de aceleración y rotación de la nave son cero, esta no se moverá. Más adelante modificaremos estos valores.

El trabajo con hilos de ejecución es especialmente delicado. Como veremos en próximos capítulos, este código nos va a ocasionar varios quebraderos de cabeza. Un primer problema es que seguirá ejecutándose aunque nuestra aplicación esté en segundo plano. Veremos cómo detener el hilo de ejecución cuando estudiemos

el ciclo de vida de una actividad. (*NOTA: Si ejecutas el programa en el terminal real, este funcionará más lentamente y consumirá más batería. Puede ser buena idea detener la aplicación.*) Un segundo problema aparecerá cuando dos hilos de ejecución traten de acceder a la misma variable a la vez. Se resolverá a continuación.



Preguntas de repaso: *ActualizaFisica()*



Ejercicio: *Introduciendo secciones críticas en Java* (synchronized)

Cuando se realiza una aplicación que ejecuta varios hilos de ejecución, hay que prestar especial cuidado a que ambos hilos puedan acceder de forma simultánea a los datos. Cuando se limitan a leer las variables, no suele haber problemas. El problema aparece cuando un hilo está modificando algún dato y justo en ese instante se pasa a ejecutar un segundo hilo que ha de leer esos datos. Este segundo hilo va a encontrar unos datos a medio modificar, lo que posiblemente cause errores en su interpretación. El método más común para resolver este problema se conoce como *exclusión mutua*, y consiste en evitar que dos hilos accedan al mismo tiempo a un recurso. En Java se consigue utilizando la palabra reservada `synchronized`.

1. Introduce la palabra reservada `synchronized` delante del método `onDraw()` y `actualizaFisica()`. De esta forma se evita que cuando `actualizaFisica()` esté modificando alguno de los valores de la nave o los asteroides en método `onDraw()` acceda a estos valores.



Nota sobre Java: *La palabra clave* `synchronized` *permite definir una sección crítica en Java. Expliquemos en qué consiste: cada vez que un hilo de ejecución (thread) entra en un método o bloque de instrucciones marcado con* `synchronized`, *se pregunta al objeto si ya hay algún otro thread que haya entrado en la sección crítica de ese objeto. La sección crítica está formada por todos los bloques de instrucciones marcados con* `synchronized`. *Si nadie ha entrado en la sección crítica, se entrará normalmente. Si ya hay otro thread dentro, entonces el thread actual se suspende y ha de esperar hasta que la sección crítica quede libre. Esto ocurrirá cuando el thread que está dentro de la sección crítica salga.*

Dos matizaciones importantes: la primera es que la sección crítica se define a nivel de objeto, no de clase. Es decir, cada objeto instanciado no influye en las secciones críticas de otros objetos. En segundo lugar, solo se define una sección crítica por objeto. Aunque se haya utilizado `synchronized` *en varios métodos, realmente solo hay una sección crítica.*

5.1.5. Ejecutar una tarea en un nuevo hilo con AsyncTask



Vídeo[tutorial]: *Ejecución en segundo plano con AsyncTask*

En Android es muy frecuente lanzar nuevos hilos. Tendremos que hacerlo siempre que exista la posibilidad de que una tarea pueda bloquear el hilo de la interfaz de usuario. Esto suele ocurrir en cálculos complejos o en accesos a la red.

Tras ver el uso de las herramientas estándares en Java para crear hilos, en este apartado veremos una clase creada en Android que nos ayudará a resolver este tipo de problemas de forma más sencilla, la clase `AsyncTask`.

Una tarea asincrónica (`AsyncTask`) permite realizar un cálculo o proceso que se ejecuta en un hilo secundario y cuyo resultado queremos que se publique en el hilo de la interfaz de usuario. Para crear una nueva tarea asíncrona puedes basarte en el siguiente esquema:

```
class MiTarea extends AsyncTask<Parametros, Progreso, Resultado> {

   @Override protected void onPreExecute() {
      …
   }
   @Override protected Resultado doInBackground(Parametros... p) {
      …
   }
   @Override protected void onProgressUpdate(Progreso... prog) {
      …
   }
   @Override protected void onPostExecute(Resultado resultado) {
      …
   }
   @Override protected void onCancelled(Resultado resultado) {
      …
   }
}
```

donde `Parametros`, `Progreso` y `Resultado` han de ser reemplazados por nombres de clases según los tipos de datos con los que trabaje la tarea.

Los cuatro métodos que podemos sobrescribir corresponden a los cuatro pasos que seguirá `AsyncTask` para ejecutar la tarea:

- `onPreExecute()`: En este método tenemos que realizar los trabajos previos a la ejecución de la tarea. Se utiliza normalmente para configurar la tarea y para mostrar en la interfaz de usuario que empieza la tarea.

- `doInBackground(Parametros...)`: Se llama cuando termina `onPreExecute()`. Es la parte más importante, donde tenemos que realizar la tarea propiamente dicha. Es el único método de los cuatro que no se ejecuta en el hilo de la interfaz de usuario. Lo va a hacer en un hilo nuevo creado para este propósito. Como hemos visto, la clase `AsyncTask` ha de ser parametrizada con tres tipos de datos. Es decir, cuando crees un `AsyncTask`, la clase `Parametros` ha de ser reemplazada por una clase concreta que será utilizada para indicar la información de entrada a la tarea. Observa como en el parámetro de este método se han añadido tres puntos después de `Parametros`. Esto significa que se puede pasar al método un número variable de parámetros de esta clase[24].
- `onProgressUpdate(Progreso...)`: Este método se utiliza para mostrar el progreso de la tarea al usuario. Se ejecuta en el hilo de la interfaz de usuario, por lo que podremos interactuar con las vistas. El progreso de una determinada tarea ha de ser controlado por nuestro código llamando al método `publishProgress(Progreso...)` desde `doInBackground()`. La clase `Progreso` es utilizada para pasar la información de progreso. Un uso frecuente es reemplazarla por `Integer` y representar el porcentaje de progreso en un valor entre el 0 y el 100.
- `onPostExecute(Resultado)`: Este método se usa para mostrar en la interfaz de usuario el resultado de la tarea. El parámetro de entrada (de la clase `Resultado`) corresponde al objeto devuelto por el método `doInBackground()`.
- `onCancelled(Resultado)`: Este método se ejecutará en el hilo principal si se cancela la ejecución de la tarea en segundo plano antes de su finalización.

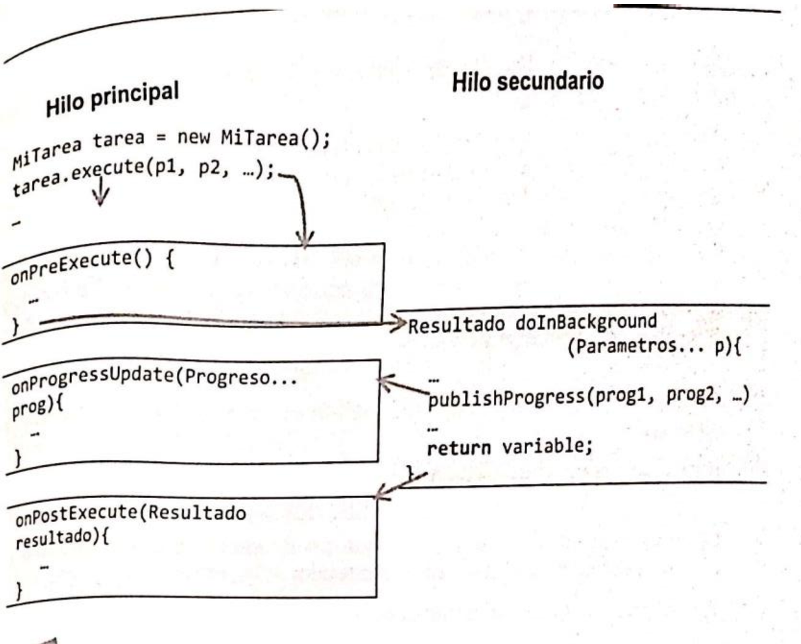
Una vez definida la clase descendiente de `AsyncTask` podremos arrancar una tarea de la siguiente forma:

```
MiTarea tarea = new MiTarea();
tarea.execute(p1, p2, p3);
```

donde `p1`, `p2`, `p3` ha de ser una lista de objetos de la clase Parametros, pudiendo introducirse un número variable de parámetros. Hay que resaltar que `execute()` es un método asíncrono. Esto significa que, tras llamarlo, se pondrá en marcha la tarea en otro hilo, pero en paralelo se continuarán ejecutando las instrucciones que hayas escrito a continuación de *execute*. El nombre de `AsyncTask` se ha puesto precisamente por este comportamiento.

En el siguiente diagrama se muestra el orden de ejecución de estos métodos:

[24] Véanse en el Anexo C los métodos con argumentos variables en número.





Ejercicio: *Crear un nuevo hilo con AsyncTask*

En este ejercicio resolveremos la operación del ejercicio anterior, pero ahora usando `AsyncTask` en lugar de `Thread`.

1. Abre el proyecto Hilos creado en el ejercicio anterior.
2. Dentro de `MainActivity` introduce el siguiente código:

```
class MiTarea extends AsyncTask<Integer, Void, Integer> {
   @Override
   protected Integer doInBackground(Integer... n) {
      return factorial(n[0]);
   }
   @Override
   protected void onPostExecute(Integer res) {
      salida.append(res + "\n");
   }
}
```

3. Cuando extendamos esta clase, hemos de comenzar decidiendo los tres tipos de datos que utilizaremos y reemplazar los tres tipos en la parametrización de la clase `AsyncTask<Parametros, Progreso, Resultado>`. En nuestra tarea necesitamos un entero como entrada, no usaremos información de progreso y devolveremos un entero. Estos tres tipos de datos solo pueden ser clases. Si

queremos utilizar un tipo simple, como `int`, tendremos que usar una clase envolvente, como `Integer`[25].

4. En el método `doInBackground()` se ha indicado como parámetro `Integer...`, de manera que se le podrán pasar una lista de enteros. Aunque en nuestro caso solo nos interesa el primero (`n[0]`), estamos obligados a sobrescribir el método exactamente como se espera, y no podemos quitar los `...`. En este método nos limitamos a calcular el factorial y devolverlo. Al terminar este método se llamará a `onPostExecute()`, pasándole como parámetro el valor devuelto. Para terminar la explicación, recuerda que el método `doInBackground()` se ejecutará en un nuevo hilo, mientras que `onPostExecute()` se ejecutará en el hilo de la interfaz de usuario.
5. Comenta las dos últimas líneas del método `calcularOperacion()` y añade las siguientes:

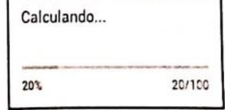
```
MiTarea tarea = new MiTarea();
tarea.execute(n);
```

La llamada a `execute()` provocará que los diferentes métodos definidos en `MiTarea` sean llamados en el orden adecuado.

6. Comprueba que funciona correctamente.

5.1.6. Mostrar un cuadro de progreso en un AsyncTask

Si estamos realizando una tarea que puede prolongarse en el tiempo, resulta muy importante mostrar al usuario cuándo empieza su progreso y cuándo termina. En el siguiente ejercicio vamos a ver un ejemplo algo más complejo de `AsyncTask`, donde usaremos la clase `ProgressDialog` para mostrar la evolución de la tarea.





Ejercicio: *Uso de un cuadro de progreso en un AsyncTask*

1. Siguiendo con el ejercicio anterior, reemplaza la clase `MiTarea` por el código:

```
class MiTarea extends AsyncTask<Integer, Integer, Integer> {
   private ProgressDialog progreso;
   @Override protected void onPreExecute() {
      progreso = new ProgressDialog(MainActivity.this);
      progreso.setProgressStyle(ProgressDialog.
                                          STYLE_HORIZONTAL);
      progreso.setMessage("Calculando...");
      progreso.setCancelable(false);
      progreso.setMax(100);
      progreso.setProgress(0);
      progreso.show();
   }
   @Override protected Integer doInBackground(Integer... n) {
      int res = 1;
      for (int i = 1; i <= n[0]; i++) {
         res *= i;
         SystemClock.sleep(1000);
         publishProgress(i*100 / n[0]);
      }
      return res;
   }
   @Override protected void onProgressUpdate(Integer... porc) {
      progreso.setProgress(porc[0]);
   }
   @Override protected void onPostExecute(Integer res) {
      progreso.dismiss();
      salida.append(res + "\n");
   }
}
```

[25] Véanse envolventes (*wrappers*) en el anexo C.

En esta nueva versión se han incluido los cuatro métodos principales de `AsyncTask`. El primero en ejecutarse será `onPreExecute()`, donde creamos un `ProgressDialog`, lo configuramos y lo mostramos. Cuando acabe, se ejecutará `doInBackground()`. En esta versión no podemos llamar simplemente a `factorial()`, dado que ahora queremos insertar en el bucle la sentencia `publishProgress(i*100/n[0])`. Lo que ocasionará una llamada a `onProgressUpdate()`, desde donde tendremos acceso a la interfaz de usuario y podremos actualizar el `ProgressDialog`. Finalmente, cuando `doInBackground()` termine se llamará a `onPostExecute()`, donde destruiremos el `ProgressDialog` y mostraremos el resultado.

2. Verifica el resultado obtenido.



Ejercicio: *Cancelando un AsyncTask*

Dado que `AsyncTask` se utiliza en tareas prolongadas, es posible que el usuario no quiera esperar a que termine o que descubramos en medio del proceso que no podemos terminar la tarea. Podemos utilizar el método `cancel()` cuando ocurra esta circunstancia. Si cancelamos una tarea, no se llamará al método `onPostExecute(Resultado)`, y en su lugar se llamará a `onCancelled()`. El siguiente ejercicio ilustra su uso:

1. En el método `onPreExecute()` del `AsyncTask` cambia el parámetro de `progreso.setCancelable(false)` a `true`.

2. A continuación de la línea que acabas de modificar, añade:

```
progreso.setOnCancelListener(new OnCancelListener() {
    @Override public void onCancel(DialogInterface dialog) {
        MiTarea.this.cancel(true);
    }
});
```

Con esto conseguimos poner un escuchador de evento al `ProgressDialog`, para que cuando sea cancelado también cancele el `AsyncTask`.

NOTA: La clase `OnCancelListener` *se define en el paquete* `android.content.DialogInterface`.

3. Dentro de `doInBackground()` añade la siguiente condición de finalización, marcada en negrita, en el bucle `for`:

```
for (int i = 1; i <= n[0] && !isCancelled(); i++) {
```

De esta forma no seguiremos realizando la tarea si nos cancelan.

4. Para terminar, añade el siguiente método en la clase `AsyncTask`:

```
@Override protected void onCancelled() {
   salida.append("cancelado\n");
}
```

5. Ejecuta la aplicación. Pulsa la tecla "retorno" para cancelar la operación.

5.1.7. El método *get()* de AsyncTask

En caso de que necesites el resultado de esta tarea para poder continuar ejecutando tu programa, puedes utilizar el siguiente método:

```
Resultado r = tarea.get();
```

Lo que hace es esperar a que termine la tarea y devuelve el resultado obtenido. Aunque parezca muy útil, este método hay que usarlo con mucho cuidado, dado que se bloquea hasta que termine la tarea, y esto es justo lo que queríamos evitar al introducir el `AsyncTask`. Veamos un ejemplo con el siguiente esquema:

Hilo principal / **Hilo secundario**

```
onClickButton() {
  tarea.execute();  ------------>  Resultado doInBackground () {
                                     ...
                   .-----------    return resultado;
     V            |              }
  r = tarea.get(); <-
}
```

Si el método `onClickButton()` se asocia a la pulsación de un botón y este se pulsa, has de tener claro que el hilo principal quedará bloqueado hasta que termine la tarea.

El método `get()` dispone de una sobrecarga alternativa que resulta muy práctica. En ella indicamos dos parámetros, donde fijamos el tiempo máximo de la tarea y en qué unidades está ese tiempo. Por ejemplo, si escribimos `get(4, TimeUnit.SECONDS)`, pasados 4 segundos se detendrá la tarea y se lanzará una excepción. Usar este método resulta interesante cuando no tenemos más remedio que bloquear el hilo de la interfaz del usuario hasta que termine una tarea.

Vemos un ejemplo de uso. Si hemos creado un `AsyncTask` que valida un usuario en nuestro servidor, podríamos usar el siguiente método:

```
public boolean onLogin(String usuario, String contrasena){
  try {
     TareaLogin tarea = new TareaLogin();
     tarea.execute(usuario, contrasena);
     return tarea.get(4, TimeUnit.SECONDS);
  } catch (TimeoutException e) {
     Toast.makeText(contexto, "Tiempo excedido al validar",
           Toast.LENGTH_LONG).show();
  } catch (CancellationException e) {
     Toast.makeText(contexto, "Error al conectar con servidor",
           Toast.LENGTH_LONG).show();
  } catch (Exception e) {
     Toast.makeText(contexto, "Error con tarea asíncrona",
           Toast.LENGTH_LONG).show();
  }
  return false;
}
```

Cuando este método sea invocado, has de tener claro que el hilo de la interfaz de usuario se va a bloquear hasta que termine la tarea. Sin embargo, en este caso particular no queremos que el usuario realice ninguna acción hasta ser validado. Por lo que no se apreciará falta de interactividad. Además, estamos limitando este bloqueo a un máximo de 4 segundos, impidiendo que aparezca el error "La aplicación no responde". En caso de producirse cualquier problema, será tratado en la sección `catch`, donde mostraremos al usuario el error que se ha producido. `TimeoutException` ocurrirá si la tarea tarda más de 4 segundos. `CancellationException` ocurrirá si el método `cancel()` es invocado dentro de la tarea, supuestamente si el servidor no responde o si la contraseña no es correcta. Pueden producirse un par de tipos de excepciones más relacionadas con hilos de ejecución. Ambas son capturadas mediante la excepción genérica `Exception`.

Hilo principal / **Hilo secundario**

```
MiTarea tarea = new MiTarea();
tarea.execute(p1, p2, …);
…
v = tarea.get(5,TimeUnit.SECONDS);

onPreExecute() {
  …
}                           Resultado doInBackground
                                    (Parametros... p){
onProgressUpdate(Progreso...    …
prog){                          publishProgress(prog1, prog2, …)
  …                             …
}                               return variable;
                            }
onPostExecute(Resultado
resultado){
  …
}
```

Al final del capítulo 10 se hace una discusión más profunda de este método en el ejercicio "Uso síncrono de AsyncTask para acceso al servicio web PHP de puntuaciones".



Preguntas de repaso: *AsyncTask*

5.2. Manejando eventos de usuario

Android captura los distintos eventos generados por el usuario de forma homogénea y se los pasa a la clase encargada de tratarlos. Por lo general, va a ser un objeto tipo `View` el que recogerá estos eventos por medio de dos técnicas alternativas: los escuchadores de eventos (*Event Listener*) y los manejadores de eventos (*Event Handler*).

5.2.1. Escuchador de eventos de la clase View

La clase `View` define varias interfaces conocidas como escuchadores de eventos o *Event Listener*. Estas interfaces definen un método *callback* al que el sistema llamará cuando se produzca una acción determinada. Cuando queramos que un objeto reaccione ante un evento de una vista, tendremos que implementar la interfaz correspondiente al evento y registrar nuestro objeto como escuchador de ese evento. Disponemos de los siguientes métodos *callback* asociados a eventos en la clase `View`:

`onClick()`

Método de la interfaz `View.OnClickListener`. Se llama cuando el usuario selecciona la vista. Se puede utilizar cualquier medio, como la pantalla táctil, las teclas de navegación.

`onLongClick()`

Método de la interfaz `View.OnLongClickListener`. Se llama cuando el usuario selecciona la vista y la mantiene seleccionada durante más de un segundo.

`onFocusChange()`

Método de la interfaz `View.OnFocusChangeListener`. Se llama cuando el usuario navega dentro o fuera de un elemento.

`onKey()`

Método de la interfaz `View.OnKeyListener`. Se llama cuando se pulsa o se suelta una tecla del dispositivo. La vista que tenga el foco en ese momento es la que generará el evento.

`onTouch()`

Método de la interfaz `View.OnTouchListener`. Se llama cuando se pulsa, se suelta o se desplaza en la pantalla táctil.

`onCreateContextMenu()`

Método de la interfaz `View.OnCreateContextMenuListener`. Se llama cuando se va a crear un menú de contexto asociado a una vista.

Existen dos alternativas para crear un escuchador de evento. La primera es crear un objeto anónimo; por ejemplo, de la interfaz `OnClickListener`:

```
protected void onCreate(Bundle savedValues) {
   ...
   Button boton = findViewById(R.id.boton);
   boton.setOnClickListener( new OnClickListener() {
      public void onClick(View v) {
         // Acciones a realizar
      }
   });
   ...
}
```

La segunda alternativa consiste en implementar la interfaz correspondiente al evento como parte de tu clase y recoger los eventos en el método *callback*. Esta alternativa es la recomendada en la documentación de Android, al tener menos gasto de memoria. A continuación se muestra un ejemplo, donde se implementa la interfaz `OnClickListener` en una clase `Activity`:

```
public class Ejemplo extends Activity implements OnClickListener{
   protected void onCreate(Bundle savedValues) {
      ...
      Button boton = findViewById(R.id.boton);
```

```
        boton.setOnClickListener(this);
    }

    public void onClick(View v) {
     // Acciones a realizar
    }
    ...
}
```

5.2.2. Manejadores de eventos

Si estás creando un descendiente de la clase View, podrás utilizar varios métodos *callback* directamente usados como manejadores de eventos (*Event Handlers*). Puedes sobrescribir uno de estos métodos cuando quieras que tu código reaccione ante un evento determinado. En esta lista se incluye:

`onKeyDown(int keyCode, KeyEvent e)` Llamado cuando se pulsa una tecla.

`onKeyUp(int keyCode, KeyEvent e)` Cuando una tecla deja de pulsarse.

`onTrackballEvent(MotionEvent me)` Llamado cuando se mueve el *trackball*.

`onTouchEvent(MotionEvent me)` Cuando se pulsa en la pantalla táctil.

`onFocusChanged(boolean obtengoFoco, int direccion, Rect prevRectanguloFoco)` Llamado cuando cambia el foco.

Es la forma más sencilla, dado que no hace falta usar o implementar la interfaz, ni registrar el escuchador. Como en nuestro ejemplo estamos creando `VistaJuego`, que es un descendiente de `View`, podremos utilizar directamente manejadores de eventos. En los siguientes apartados mostraremos varios ejemplos.



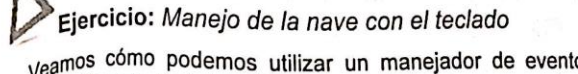
Vídeo[tutorial]: *Escuchadores y manejadores de eventos*



Preguntas de repaso: *Manejo de eventos*

5.3. El teclado

Aunque la mayoría de los terminales Android no tienen teclado físico, podemos encontrar algunos que sí disponen de él (por ejemplo, el mando a distancia de un *set-top box* o un *netPC*). Por lo tanto, resulta interesante aprender a gestionar los eventos procedentes del teclado. Su manejo se ilustra en el siguiente ejercicio.



Ejercicio: *Manejo de la nave con el teclado*

Veamos cómo podemos utilizar un manejador de eventos de teclado para maniobrar la nave de Asteroides:

1. Abre el proyecto Asteroides.
2. Inserta este código en la clase `VistaJuego`:

```
@Override
public boolean onKeyDown(int codigoTecla, KeyEvent evento) {
   super.onKeyDown(codigoTecla, evento);
    // Suponemos que vamos a procesar la pulsación
   boolean procesada = true;
   switch (codigoTecla) {
   case KeyEvent.KEYCODE_DPAD_UP:
      aceleracionNave = +PASO_ACELERACION_NAVE;
      break;
   case KeyEvent.KEYCODE_DPAD_LEFT:
      giroNave = -PASO_GIRO_NAVE;
      break;
   case KeyEvent.KEYCODE_DPAD_RIGHT:
      giroNave = +PASO_GIRO_NAVE;
      break;
   case KeyEvent.KEYCODE_DPAD_CENTER:
   case KeyEvent.KEYCODE_ENTER:
      activaMisil();
      break;
   default:
      // Si estamos aquí, no hay pulsación que nos interese
      procesada = false;
      break;
   }
   return procesada;
}
```

Cada vez que se pulse una tecla se realizará una llamada al método `onKeyDown()` con los siguientes parámetros: el primero es un entero que nos identifica el código de la tecla pulsada; el segundo es de la clase `KeyEvent` y nos permite obtener información adicional sobre el evento (por ejemplo, cuándo se produjo). Este método ha de devolver un valor booleano, verdadero, si consideramos que la pulsación ha sido procesada por nuestro código, y falso, si queremos que otro manejador de eventos, siguiente al nuestro, reciba la pulsación.

3. Antes de ponerlo en marcha, comenta la llamada a `activaMisil()`, dado que esta función aún no está implementada.
4. Verifica si funciona correctamente.

NOTA: Para poder recoger eventos de teclado desde una vista es necesario que esta tenga el foco, y para que esto sea posible verifica que tiene la propiedad `focusable="true"`.



Práctica: *Manejo de la nave con el teclado*

El ejercicio anterior no funciona de forma satisfactoria. Cuando pulsamos una tecla para girar, la nave se pone a girar, pero ya no hay manera de pararla. El manejador de eventos `onKeyDown` solo se activa cuando se pulsa una tecla, pero no cuando se suelta. Trata de escribir el manejador de eventos `onKeyUp` para que la nave atienda a las órdenes de forma correcta. Puedes partir del siguiente código:

```
@Override public boolean onKeyUp(int codigoTecla, KeyEvent evento) {
   super.onKeyUp(codigoTecla, evento);
   // Suponemos que vamos a procesar la pulsación
   boolean procesada = true;
   ...
   return procesada;
}
```



Solución: A continuación se muestra una posible solución al ejercicio:

```
@Override public boolean onKeyUp(int codigoTecla, KeyEvent evento) {
   super.onKeyUp(codigoTecla, evento);
   // Suponemos que vamos a procesar la pulsación
   boolean procesada = true;
   switch (codigoTecla) {
   case KeyEvent.KEYCODE_DPAD_UP:
      aceleracionNave = 0;
      break;
   case KeyEvent.KEYCODE_DPAD_LEFT:
   case KeyEvent.KEYCODE_DPAD_RIGHT:
      giroNave = 0;
      break;
   default:
      // Si estamos aquí, no hay pulsación que nos interese
      procesada = false;
      break;
   }
   return procesada;
}
```


5.4. La pantalla táctil

Los dispositivos Android suelen incorporar una pantalla táctil, que es utilizada como método principal de entrada. El uso más importante de la pantalla táctil es como sustituto del ratón de un ordenador de sobremesa. De esta forma podemos seleccionar, arrastrar y soltar cualquier elemento de la pantalla de forma sencilla. No obstante, el uso de este dispositivo no acaba aquí. Suele utilizarse en sustitución del teclado en aquellos dispositivos que no disponen de teclado físico. También puede ser utilizada como entrada de un videojuego, como se verá en este apartado.

El manejo básico de la pantalla táctil pasa por definir el método `onTouchEvent` en una clase `View` (o implementar la interfaz `OnTouchListener` en otras clases). Este método nos pasará en un parámetro un objeto de la clase `MotionEvent` con información sobre el evento. Los métodos más interesantes de la clase `MotionEvent` se indican a continuación:

`getAction()` Tipo de acción realizada. Puede ser `ACTION_DOWN`, `ACTION_MOVE`, `ACTION_UP` o `ACTION_CANCEL`; y algunos más para gestionar pantallas multi-touch.

`getX()`, `getY()` Posición de la pulsación.

`getDownTime()` Tiempo en ms en que el usuario presionó por primera vez en una cadena de eventos de posición.

`getEventTime()` Tiempo en ms del evento actual.

`getPressure()` Estima la presión de la pulsación. El valor 0 es el mínimo, el valor 1 representa una pulsación normal.

`getSize()` Valor escalado en 0 y 1 que estima el grosor de la pulsación.

A estos métodos les podemos pasar como parámetro un índice de puntero para indicar al sistema sobre cuál de los distintos punteros (dedo, marcador, ...) estamos consultando. Esto nos permite trabajar con pantallas *multitouch*, donde se puede detectar la pulsación de varios dedos simultáneamente. Lo estudiaremos en el siguiente apartado.



Ejercicio: *Uso de la pantalla táctil*

En este ejercicio se mostrará cómo podemos capturar los eventos procedentes de la pantalla táctil. También se aprovechará para repasar otros conceptos, como creación de *layouts* y herramientas de revisión de código.

1. Crea un nuevo proyecto con nombre *PantallaTactil* y tipo *Empty Activity*.
2. Modifica el *layout activity_main.xml* para que tenga una apariencia similar a la siguiente. De esta forma practicarás la creación de *layouts*. A la derecha se muestra la estructura de vistas que contiene.



A continuación se muestra una posible solución:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/TextViewEntrada"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:text="Pulsa sobre esta vista"
        android:gravity="center"
        android:background="#0000FF"
        android:layout_margin="2mm"
        android:textSize="10pt"/>
    <ScrollView
        android:id="@+id/scrollView1"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1" >
        <TextView
            android:id="@+id/TextViewSalida"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:text="Pulsaciones:"/>
    </ScrollView>
</LinearLayout>
```

3. Introduce las siguientes dos líneas al final del método `onCreate()`:

```
TextView entrada = findViewById(R.id.TextViewEntrada);
entrada.setOnTouchListener(this);
```

4. Pulsa ***Alt-Intro*** para añadir los *imports*.

5. Observa como el método `setOnTouchListener` está marcado como erróneo. Si pones el cursor encima, te indicará que el parámetro de este método (`this`) es de la clase `MainActivity`, y es necesario que sea de tipo `OnTouchListener`.

   Para evitar el error te mostrará una lista de posibles soluciones. Selecciona la última: "`Let 'MainActivity' implement 'OnTouchListener'`", de esta forma implementaremos esta interfaz y nuestra clase podrá ser considerada de este tipo. La declaración de la clase cambiará a:

```
public class MainActivity extends Activity implements OnTouchListener {
```

6. Se ha solucionado el problema anterior, pero ha aparecido otro: la `MainActivity` está marcada como errónea. El problema consiste en que estamos diciendo que implementamos la interfaz `OnTouchListener`, pero no hemos implementado el método de esta interfaz.

   Para evitar el error selecciona en la lista de posibles soluciones: "`Add unimplemented methods`", de esta forma se añadirán todos los métodos necesarios de esta interfaz. La declaración de la clase cambiará a:

```
public boolean onTouch(View arg0, MotionEvent arg1) {
    // TODO Auto-generated method stub
    return false;
}
```

7. Reemplaza el nombre de los parámetros por otros más expresivos. Por ejemplo: `arg0` por `vista` y `arg1` por `evento`.
8. Observa como este método ha de devolver un parámetro. Actualmente es `false`, que significa que no nos hemos hecho cargo de la pulsación: el sistema seguirá pasando este evento a otras vistas. En este caso, el `LinearLayout` que contiene la vista. Cámbialo a `true`, para que el sistema no siga propagando este evento.
9. Reemplaza la línea "`// TODO Auto-generated method stub`" por:

```
TextView salida = findViewById(R.id.TextViewSalida);
salida.append(evento.toString()+"\n");
```

10. Ejecuta el proyecto y verifica el resultado.

    `action = MotionEvent.ACTION_DOWN = 0` significa que se ha pulsado sobre la pantalla, `action = MotionEvent.ACTION_UP = 1` significa que se ha soltado y `action = MotionEvent.ACTION_MOVE = 2`, que se está desplazando el dedo.

11. Verifica el resultado en un dispositivo real.

    No todas las pantallas táctiles soportan los métodos `getPressure()` y `getSize()`. Prueba con tu terminal si lo soporta, y en tal caso observa el rango de valores que obtienes [valor_min, valor_max] para el valor devuelto.

5.4.1. Manejo de la pantalla táctil *multi-touch*

Las pantallas táctiles actuales tienen la posibilidad de indicar la posición de varios punteros sobre la pantalla a un mismo tiempo.

Un objeto `MotionEvent` contiene información de todos estos punteros. Un puntero estará activo desde que se pulsa sobre la pantalla hasta que se deja de presionar. El número de punteros activos puede consultarse llamando al método `getPointerCount()`. Cada puntero tiene un *id* para identificarlo que se asigna cuando se produce la primera pulsación.

La clase `MotionEvent` dispone de la siguiente lista de constantes para identificar acciones posibles que se adaptan a *multi-touch*:

`ACTION_DOWN` – Se pulsa en la pantalla sin que haya otro puntero activo.

`ACTION_UP` – Se deja de presionar el último puntero activo.

`ACTION_MOVE` – Cualquiera de los punteros activos se desplaza.

`ACTION_CANCEL` – Se cancela un *gesture*.

`ACTION_OUTSIDE` – El puntero se sale de la vista.

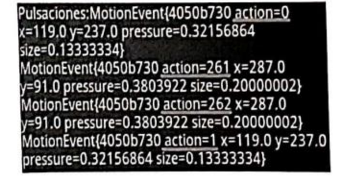
`ACTION_POINTER_DOWN` – Se pulsa un nuevo puntero distinto al primero.

`ACTION_POINTER_UP` – Se deja de presionar un puntero pero no es el último.



Ejercicio: *Uso de la pantalla táctil* multi-touch

1. Ejecuta el ejercicio anterior en un dispositivo real con capacidad de *multi-touch* (si no dispones de uno, te será imposible realizar este ejercicio).
2. Pulsa simultáneamente con dos dedos en la pantalla. Si lo haces sin desplazar los dedos recibirás 4 eventos: los dos primeros por las pulsaciones de cada dedo y los dos siguientes por levantarlos. El resultado puede ser similar al siguiente:



```
Pulsaciones:MotionEvent{4050b730 action=0
x=119.0 y=237.0 pressure=0.32156864
size=0.13333334}
MotionEvent{4050b730 action=261 x=287.0
y=91.0 pressure=0.3803922 size=0.20000002}
MotionEvent{4050b730 action=262 x=287.0
y=91.0 pressure=0.3803922 size=0.20000002}
MotionEvent{4050b730 action=1 x=119.0 y=237.0
pressure=0.32156864 size=0.13333334}
```

Como puedes ver, cuando hay más de un puntero en pantalla, la acción resulta compleja de interpretar. A continuación veremos cómo hacerlo.

3. Reemplaza la siguiente línea del método `onTouch()`:

```
salida.append(evento.toString()+"\n");
```

por:

```
String acciones[] = { "ACTION_DOWN", "ACTION_UP", "ACTION_MOVE",
        "ACTION_CANCEL", "ACTION_OUTSIDE", "ACTION_POINTER_DOWN",
        "ACTION_POINTER_UP" };
int accion = evento.getAction();
int codigoAccion = accion & MotionEvent.ACTION_MASK;
```

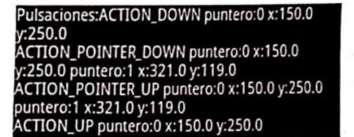


```
salida.append(acciones[codigoAccion]);
for (int i = 0; i < evento.getPointerCount(); i++) {
  salida.append(" puntero:" + evento.getPointerId(i) +
          " x:" + evento.getX(i) + " y:" + evento.getY(i));
}
salida.append("\n");
```

Para visualizar cada posible acción hemos creado un *array* con sus nombres. A continuación averiguamos la acción en la variable `accion`. Esta nueva acción la ha podido hacer cualquier puntero de los activos o uno nuevo. En esta variable se codifican simultáneamente el código de la acción (los 8 bits menos significativos) y el índice de puntero que la ocasiona (los 8 bits siguientes). Para obtener esta información por separado puedes utilizar el siguiente código:

```
int codigoAccion = accion & MotionEvent.ACTION_MASK;
int iPuntero = (accion & MotionEvent.ACTION_POINTER_INDEX_MASK)
                    >> MotionEvent.ACTION_POINTER_INDEX_SHIFT;
```

4. Una vez obtenido el código de la acción, mostramos su nombre en la vista `salida`. Luego hacemos un bucle para mostrar información de todos los punteros activos. El método `getPointerCount()` nos permite averiguar su número. Vamos a recorrer los punteros activos con la variable `i`. Al principio de este apartado hemos visto una serie de métodos para obtener información sobre el puntero (`getX()`, `getSize()`, etc.). Estos métodos dan información sobre el primer puntero activo que se pulsó, pero también indican un índice de puntero (`getX(i)`, `getSize(i)`, etc.) para obtener información sobre el resto de los punteros activos.
5. El método `getPointerId(int indice)` nos permite averiguar el identificador del puntero. No hay que confundir el índice de puntero con su identificador. El índice se asigna en función del orden en que fueron pulsados. El índice cero siempre es el más antiguo, el índice uno es el siguiente que se pulsó, etc. El índice de un puntero decrece a medida que los punteros anteriores a él dejan de estar activos. Por el contrario, el identificador de un puntero se asigna cuando se crea y permanece constante durante toda su vida. Nos será muy útil para seguir la pista de un determinado puntero. El método `findPointerIndex(int id)` nos permite averiguar el índice de un puntero a partir de su identificador.
6. Ejecuta de nuevo el proyecto y vuelve a pulsar con dos dedos. El resultado ha de ser similar al siguiente:



```
Pulsaciones:ACTION_DOWN puntero:0 x:150.0
y:250.0
ACTION_POINTER_DOWN puntero:0 x:150.0
y:250.0 puntero:1 x:321.0 y:119.0
ACTION_POINTER_UP puntero:0 x:150.0 y:250.0
puntero:1 x:321.0 y:119.0
ACTION_UP puntero:0 x:150.0 y:250.0
```

7. Modifica el programa para que además se muestre en cada evento el índice de puntero que lo ocasionó.
8. Prueba con otras combinaciones de pulsaciones e investiga la relación entre el índice y el *id* de puntero.

5.4.2. Manejo de la nave con la pantalla táctil



Ejercicio: *Manejo de la nave con la pantalla táctil*

Veamos cómo podemos utilizar un manejador de eventos de la pantalla táctil para maniobrar la nave de Asteroides. El código que se muestra permite manejar la nave de la siguiente forma: un desplazamiento del dedo horizontal hace girar la nave, un desplazamiento vertical produce una aceleración y si al soltar la pulsación no hay movimiento, se provoca un disparo.

1. Abre el proyecto Asteroides.
2. Inserta este código en la clase `VistaJuego`:

```
private float mX=0, mY=0;
private boolean disparo=false;

@Override
public boolean onTouchEvent (MotionEvent event) {
   super.onTouchEvent(event);
   float x = event.getX();
   float y = event.getY();
   switch (event.getAction()) {
   case MotionEvent.ACTION_DOWN:
      disparo=true;
      break;
   case MotionEvent.ACTION_MOVE:
      float dx = Math.abs(x - mX);
      float dy = Math.abs(y - mY);
      if (dy<6 && dx>6){
         giroNave = Math.round((x - mX) / 2);
         disparo = false;
      } else if (dx<6 && dy>6){
         aceleracionNave = Math.round((mY - y) / 25);
         disparo = false;
      }
      break;
   case MotionEvent.ACTION_UP:
      giroNave = 0;
      aceleracionNave = 0;
      if (disparo){
      activaMisil();
      }
      break;
   }
   mX=x; mY=y;
   return true;
}
```

Las variables globales `mX` y `mY` se utilizarán para recordar las coordenadas del último evento. Comparándolas con las actuales (`x`, `y`) podremos verificar si se trata de un desplazamiento horizontal o vertical. Por otra parte, la variable `disparo` se activa

cada vez que comienza una pulsación (`ACTION_DOWN`). Si esta pulsación se continua con un desplazamiento horizontal o vertical, `disparo` se desactiva. Si, por el contrario, se levanta el dedo (`ACTION_UP`) sin haberse producido estos desplazamientos, `disparo` no estará desactivado y se llamará a `activaMisil()`.

3. Antes de ponerlo en marcha, comenta la llamada a `activaMisil()`, dado que esta función aún no está implementada.
4. Verifica si funciona correctamente. Desplaza el dedo sobre la pantalla y comprueba el resultado.
5. Modifica los parámetros de ajuste (`<6`, `>6`, `/2`, `/25`), para que se adapten de forma adecuada a tu terminal.
6. En el juego original podíamos acelerar, pero no decelerar. Si queríamos detener la nave, teníamos que dar un giro de 180 grados y acelerar lo justo. Modifica el código anterior para que no sea posible decelerar.

5.5. Los sensores

Bajo la denominación de sensores se engloba un conjunto de dispositivos con los que podremos obtener información del mundo exterior (en este conjunto no se incluye la cámara, el micrófono ni el GPS). Como se verá en este apartado, todos los sensores se manipulan de forma homogénea. Son los dispositivos de entrada más novedosos que incorpora Android y con ellos podremos implementar formas atractivas de interacción con el usuario.



Vídeo[tutorial]: *Sensores en dispositivos móviles*

Android permite acceder a los sensores internos del dispositivo a través de las clases `Sensor`, `SensorEvent`, `SensorManager` y la interfaz `SensorEventListener`, del paquete `android.hardware`. La clase Sensor acepta varios tipos de sensores. Aunque los sensores disponibles varían en función del dispositivo utilizado. La siguiente tabla muestra los tipos de sensores disponibles:

Tipo CONSTANTE	**Qué mide**	**Dim.**	**Unid.**	**Desde API**
acelerómetro TYPE_ACCELEROMETER	Aceleraciones por gravedad y cambios de movimiento.	3	m/s²	3
gravedad TYPE_GRAVITY	Aceleración debida a la gravedad.	3	m/s²	9
acelerómetro lineal TYPE_LINEAR_ACCELERATION	Aceleraciones sin tener en cuenta la gravedad.	3	m/s²	9
giroscopio TYPE_GYROSCOPE	Cambios de rotación.	3	rad/s	3
vector de rotación TYPE_ROTATION_VECTOR	Detectar rotaciones.	3	adimensional	9

Tipo CONSTANTE	Qué mide	Dim.	Unid.	Desde API
Orientación TYPE_ORIENTATION	Dirección a la que apunta el dispositivo (obsoleto desde API 8[26]).	3	grado	3
campo magnético TYPE_MAGNETIC_FIELD	Brújula, detectar campos magnéticos.	3	μT	3
luz ambiental TYPE_LIGHT	Luz ambiente (útil para ajustar iluminación de pantalla).	1	lx	3
proximidad TYPE_PROXIMITY	Distancia a un objeto (para saber si teléfono está en oreja).	1	cm	3
presión atmosférica TYPE_PRESSURE	Barómetro. Indirectamente podemos obtener un altímetro,	1	hPa	3
temperatura ambiental TYPE_AMBIENT_TEMPERATURE	Temperatura del aire.	1	°C	14
temperatura interna TYPE_TEMPERATURE	Para evitar sobrecalentamientos (obsoleto desde API 14).	1	°C	3
humedad relativa TYPE_RELATIVE_HUMIDITY	Punto de rocío, humedad absoluta y relativa.	1	%	14
movimiento significativo TYPE_SIGNIFICANT_MOTION	Detecta que el dispositivo ha sido movido.	triger	-	18
detector de pasos TYPE_STEP_DETECTOR	Detecta que el usuario del dispositivo da un paso.	triger	-	19
contador de pasos TYPE_STEP_COUNTER	Número de pasos realizados desde el último reinicio.	1	paso	19
frecuencia cardiac TYPE_HEART_RATE	Monitor cardíaco en pulsos por minuto.	1	rpm	20

Puedes instalar la aplicación InfoSensores[27] para conocer los sensores disponibles en tu dispositivo.



Ejercicio: *Listar los sensores del dispositivo*

No todos los dispositivos disponen de los mismos sensores. Por lo tanto, la primera tarea consiste en averiguar los sensores disponibles.

1. Crea un nuevo proyecto con nombre *Sensores* y tipo *Empty Activity.*

[26] La dirección se obtiene combinando la información obtenida del acelerómetro y el campo magnético. Se ha marcado como obsoleto, porque no soportaba cambios de orientación horizontal/vertical. No obstante, puedes seguir utilizándolo. Para evitar, evitar estos problemas puedes trabajar con una orientación fija (como hemos hecho en la actividad Juego). Obtener la dirección a partir del acelerómetro y el campo magnético es complejo.

[27] https://play.google.com/store/apps/details?id=com.gonpuga.infosensores

2. Añade la siguiente propiedad al `TextView` de *res/layout/activity_main.xml*:

```
android:id="@+id/salida"
```

3. Inserta este código en la actividad principal:

```
public class MainActivity extends Activity {
   private TextView salida;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout. activity_main);
      salida = findViewById(R.id.salida);
      SensorManager sensorManager = (SensorManager)
                              getSystemService(SENSOR_SERVICE);
      List<Sensor> listaSensores = sensorManager.
                       getSensorList(Sensor.TYPE_ALL);
      for(Sensor sensor: listaSensores) {
         log(sensor.getName());
      }
   }

   private void log(String string) {
      salida.append(string + "\n");
   }
}
```

El método comienza indicando el *layout* de la actividad y obteniendo el `TextView` `salida`, donde mostraremos los resultados. A continuación vamos a utilizar el método `getSystemService` para solicitar al sistema servicios específicos. Este método pertenece a la clase `Context` (como somos `Activity`, también somos `Context`) y será muy utilizados para acceder a una gran cantidad de servicios del sistema. Al indicar como parámetro `SENSOR_SERVICE`, indicamos que queremos utilizar los sensores. Lo haremos a través del objeto `sensorManager`. En primer lugar llamamos al método `getSensorList()` del objeto para que nos de `listaSensores`, una lista de objetos `Sensor`. La siguiente línea recorre todos los elementos de esta lista parar llamar a su método `getName()` para mostrar el nombre de sensor.

4. Ejecuta el programa. Esta es una lista de los valores devueltos por el código anterior ejecutándose en un Samsung Galaxy S3:

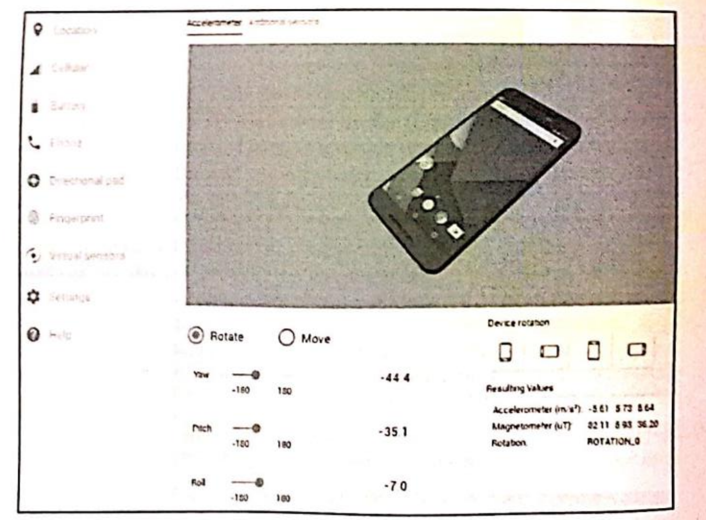
```
ALPS 3-axis Magnetic Field Sensor
MPU-6050 Accelerometer
MPU-6050 Gyroscope
GP2A Proximity Sensor
Rotation Vector Sensor
Gravity Sensor
Linear Acceleration Sensor
Orientation Sensor
Corrected Gyroscope Sensor
```

Como hemos visto, la clase `Sensor` nos permite manipular los sensores. A continuación se enumeran los métodos públicos de la clase `Sensor`:

`public float getMaximumRange()`	Rango máximo en las unidades del sensor.
`public float getMinDelay()`	Tiempo mínimo entre dos eventos (en microsegundos).
`public String getName()`	Nombre del sensor.
`public float getPower()`	Potencia (mA) usada por el sensor mientras está en uso.
`public float getResolution ()`	Resolución en las unidades del sensor.
`public int getType()`	Tipo genérico del sensor.
`Public String getVendor()`	Fabricante del sensor.
`public int getVersion()`	Versión del sensor.

La clase `SensorManager` tiene, además, tres métodos (`getInclination`, `getOrientation` y `getRotationMatrix`), usados para calcular transformaciones de coordenadas.

En Android Studio, los AVD incorporan la posibilidad de emular casi todos los sensores. En la siguiente captura se muestra la emulación del acelerómetro:





Ejercicio: *Acceso a los datos del sensor*

Veamos ahora cómo obtener la lectura de cada uno de los sensores.

1. Copia el siguiente código al final de `onCreate()` de la actividad anterior:


```
listaSensores = sensorManager.getSensorList(Sensor.TYPE_ORIENTATION);
if (!listaSensores.isEmpty()) {
  Sensor orientationSensor = listaSensores.get(0);
  sensorManager.registerListener(this, orientationSensor,
                              SensorManager.SENSOR_DELAY_UI);}
listaSensores = sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
if (!listaSensores.isEmpty()) {
  Sensor acelerometerSensor = listaSensores.get(0);
  sensorManager.registerListener(this, acelerometerSensor,
                              SensorManager.SENSOR_DELAY_UI);}
listaSensores = sensorManager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
if (!listaSensores.isEmpty()) {
  Sensor magneticSensor = listaSensores.get(0);
  sensorManager.registerListener(this, magneticSensor,
                              SensorManager.SENSOR_DELAY_UI);}
listaSensores = sensorManager.getSensorList(Sensor.TYPE_PROXIMITY);
if (!listaSensores.isEmpty()) {
  Sensor proximidadSensor = listaSensores.get(0);
  sensorManager.registerListener(this, proximidadSensor,
                              SensorManager.SENSOR_DELAY_UI);}
```

Comenzamos consultando si disponemos de un sensor de orientación. Para ello pedimos al sistema que nos dé todos los sensores de este tipo llamando a `getSensorList(Sensor.TYPE_ACCELEROMETER)`. Si la lista no está vacía obtenemos el primer elemento (posición 0). Es necesario registrar cada tipo de sensor por separado para poder obtener información de él. El método `registerListener()` toma como primer parámetro un objeto que implemente la interfaz `SensorEventListener`. Veremos a continuación cómo se implementa esta interfaz (se indica `this` porque la clase que estamos definiendo implementará esta interfaz para recoger eventos de sensores). El segundo parámetro es el sensor que estamos registrando. Y el tercero indica al sistema con qué frecuencia nos gustaría recibir actualizaciones del sensor. Acepta cuatro posibles valores. De menor a mayor frecuencia tenemos: `SENSOR_DELAY_NORMAL`, `SENSOR_DELAY_UI`, `SENSOR_DELAY_GAME` y `SENSOR_DELAY_FASTEST`. Esta indicación sirve para que el sistema estime cuánta atención necesitan los sensores, pero no garantiza una frecuencia concreta.

2. Para que nuestra clase implemente la interfaz que hemos comentado, añade a la declaración de la clase:

```
implements SensorEventListener
```

3. Para recibir los datos de los sensores, tenemos que implementar dos métodos de la interfaz `SensorEventListener`:

```
@Override
public void onAccuracyChanged(Sensor sensor, int precision) {}

@Override
public void onSensorChanged(SensorEvent evento) {
  switch(evento.sensor.getType()) {
    case Sensor.TYPE_ORIENTATION:
```

```
         for (int i=0 ; i<3 ; i++) {
            log("Orientación "+i+": "+evento.values[i]);
         }
         break;
      case Sensor.TYPE_ACCELEROMETER:
         for (int i=0 ; i<3 ; i++) {
            log("Acelerómetro "+i+": "+evento.values[i]);
         }
         break;
      case Sensor.TYPE_MAGNETIC_FIELD:
         for (int i=0 ; i<3 ; i++) {
            log("Magnetismo "+i+": "+evento.values[i]);
         }
         break;
      default:
         for (int i=0 ; i<evento.values.length ; i++) {
            log("Proximidad"+i+": "+evento.values[i]);
         }
      }
   }
}
```

Cuando implementamos una interfaz, estamos obligados a implementar todos sus métodos. En este caso son dos. Para `onAccuracyChanged` no queremos ninguna acción específica, pero lo tenemos que incluir. Cuando un sensor cambie se llamará al método `onSensorChanged`. Aquí comprobamos qué sensor ha causado la llamada y mostramos los datos. Los posibles valores devueltos dependen del tipo de sensor. Para más información, véase la documentación de la clase `SensorEvent`[28].

4. Verifica que el programa funciona correctamente.

5.5.1. Un programa que muestra los sensores disponibles y sus valores en tiempo real

La aplicación realizada en el ejercicio anterior resulta algo difícil de utilizar. En primer lugar, presupone que se dispone de cuatro sensores, cosa que no será cierta en muchos dispositivos. Además, los datos de los sensores cambian demasiado rápido para poder leerlos en una lista. El siguiente ejercicio es similar a los ejemplos anteriores, pero ahora se muestran en la pantalla los valores actuales de todos los sensores del dispositivo. Además, es un buen ejemplo para ilustrar cómo crear vistas dinámicamente desde código.



Ejercicio: *Creación de una vista desde código para mostrar los datos de los sensores*

1. Crea un nuevo proyecto con nombre *Sensores2* y tipo *Empty Activity*.

[28] http://developer.android.com/reference/android/hardware/SensorEvent.html

2. Abre el *layout activity_main.xml* y reemplaza su código por el siguiente:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/raiz"
    android:orientation="vertical">
</LinearLayout>
```

3. Reemplaza el código de la actividad por el siguiente:

```
public class MainActivity extends Activity implements SensorEventListener{

private List<Sensor> listaSensores;
private TextView aTextView[][] = new TextView[50][3];

@Override public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout. activity_main);
   LinearLayout raiz = findViewById(R.id.raiz);
   SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
   listaSensores = sm.getSensorList(Sensor.TYPE_ALL);
   int n = 0;
   for (Sensor sensor : listaSensores) {
      TextView mTextView = new TextView(this);
      mTextView.setText(sensor.getName());
      raiz.addView(mTextView);
      LinearLayout nLinearLayout = new LinearLayout(this);
      raiz.addView(nLinearLayout);
      for (int i = 0; i < 3; i++) {
         aTextView[n][i] = new TextView(this);
         aTextView[n][i].setText("?");
         aTextView[n][i].setWidth(87);
      }
      TextView xTextView = new TextView(this);
      xTextView.setText(" X: ");
      nLinearLayout.addView(xTextView);
      nLinearLayout.addView(aTextView[n][0]);
      TextView yTextView = new TextView(this);
      yTextView.setText(" Y: ");
      nLinearLayout.addView(yTextView);
      nLinearLayout.addView(aTextView[n][1]);
      TextView zTextView = new TextView(this);
      zTextView.setText(" Z: ");
      nLinearLayout.addView(zTextView);
      nLinearLayout.addView(aTextView[n][2]);
      sm.registerListener(this, sensor, SensorManager.SENSOR_DELAY_UI);
      n++;
   }
}

@Override public void onAccuracyChanged(Sensor sensor, int accuracy) {}
```

```
@Override public void onSensorChanged(SensorEvent event) {
    synchronized (this) {
        int n = 0;
        for (Sensor sensor: listaSensores) {
            if (event.sensor == sensor) {
                for (int i=0; i<event.values.length && i<3; i++) {
                    aTextView[n][i].setText(Float.toString(event.values[i]));
                }
            }
            n++;
        }
    }
}
```

Esta actividad utiliza el *layout* creado desde XML, que básicamente es un `LinearLayout` vacío. Desde Java accederemos a este *layout* con el objeto `raiz`. Al *layout* se le irán añadiendo una serie de vistas según los sensores encontrados en el dispositivo. Por cada sensor se añade: un `TextView` con el nombre del sensor; un `LinearLayout` de tipo horizontal[29] para contener, a su vez, un `TextView` con "X:"; un `TextView` con el valor del sensor en el eje X; un `TextView` con "Y"; un `TextView` con el valor del sensor en el eje Y; un `TextView` con "Z:", y un `TextView` con el valor del sensor en el eje Z. Las referencias a los `TextView` donde se visualizarán los valores de los sensores se almacenan en el *array* `aTextView[][]`, donde el primer índice identifica el número de sensor y el segundo, la dimensión X, Y o Z.

En el método `onSensorChanged()` se hace un bucle para localizar el índice del sensor que ha cambiado y se modifican los `TextView` correspondientes al sensor con los valores leídos.

NOTA: No todos los sensores tienen tres dimensiones. Por ejemplo, en el caso del sensor de proximidad solo se cambiará en el valor de X. Mientras que otros, como los vectores de rotación, pueden devolver hasta 5 valores.

4. Verifica sobre un dispositivo real que el programa funciona correctamente.

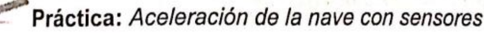
5.5.2. Utilización de los sensores en Asteroides

A continuación proponemos una serie de ejercicios y prácticas para manejar la nave de Asteroides utilizando los sensores.



Ejercicio: *Manejo de la nave con el sensor de orientación*

1. En primer lugar, implementa la interfaz `SensorEventListener`.



```
public class VistaJuego extends View implements SensorEventListener {
```

[29] La orientación por defecto en un `LinearLayout` es horizontal.

2. En el constructor registra el sensor e indica que nuestro objeto recogerá la llamada *callback*:

```
SensorManager mSensorManager = context.getSystemService(
                                        Context.SENSOR_SERVICE);
List<Sensor> listSensors = mSensorManager.getSensorList(
                                        Sensor.TYPE_ORIENTATION);
if (!listSensors.isEmpty()) {
   Sensor orientationSensor = listSensors.get(0);
   mSensorManager.registerListener(this, orientationSensor,
                                        SensorManager.SENSOR_DELAY_GAME);
}
```

NOTA: Android Studio informará de que el tipo de sensor `TYPE_ORIENTATION` *está obsoleto. No obstante, la aplicación podrá compilarse y funcionará correctamente.*

3. Añade los dos métodos que implementan la interfaz `SensorEventListener`:

```
@Override public void onAccuracyChanged(Sensor sensor, int accuracy) {}
private boolean hayValorInicial = false;
private float valorInicial;
@Override public void onSensorChanged(SensorEvent event) {
   float valor = event.values[1];
   if (!hayValorInicial){
      valorInicial = valor;
      hayValorInicial = true;
   }
   giroNave=(int) (valor-valorInicial)/3 ;
}
```

4. Prueba la aplicación. Has de tener cuidado de que el terminal esté en una posición cómoda al entrar en la actividad `Juego`, dado que el movimiento de la nave se obtiene con la diferencia de la posición del terminal con respecto a la posición inicial.



Práctica: *Manejo de la nave con sensor de aceleración*

Modifica el ejemplo anterior para utilizar el sensor de aceleración en lugar del de orientación. Gracias a la fuerza de gravedad que la Tierra ejerce sobre el terminal podremos saber si este está horizontal. En caso de que la nave esté horizontal (o casi) no ha de girar, pero cuando el terminal se incline, la nave ha de girar proporcionalmente a esa inclinación. Utiliza los programas anteriores para descubrir qué eje (*x*, *y* o *z*) es el que te interesa y el rango de valores que proporciona.

Práctica: *Aceleración de la nave con sensores*

¿Te animarías a controlar la aceleración de la nave con los sensores? Ten cuidado de que no acelere con mucha facilidad: este juego resulta muy difícil cuando

la nave está en movimiento. Puede ser una buena idea que permitas también decelerar la nave.



Práctica: *Configuración de tipo de entrada en preferencias*

Todos los controles de la nave (teclado, pantalla táctil y sensores) están activados simultáneamente. El teclado y la pantalla táctil no interfieren cuando el usuario no quiere utilizarlos. Sin embargo, la activación de los sensores sí que molestará a los usuarios que no quieran utilizar este método de entrada.

1. Crea nuevas entradas en la configuración para activar o desactivar cada tipo de entrada (o al menos la de los sensores).
2. Modifica el código anterior para que se desactiven las entradas que el usuario no haya seleccionado.

5.5.3. Restricciones al uso de sensores en segundo plano en Android 9

Android 9 presenta características nuevas para mejorar la administración de energía de los dispositivos. Estos cambios, sumados a características que ya estaban disponibles antes de Android 9, garantizan que se proporcionen los recursos de sistema a las aplicaciones que más los necesiten. Una de las nuevas características es que Android 9 limita la capacidad de las aplicaciones en segundo plano para acceder a entradas del usuario y datos de sensores.

Si tu aplicación se ejecuta en segundo plano en un dispositivo con Android 9, el sistema aplica a esta las siguientes restricciones:

- Tu aplicación no puede acceder al micrófono ni a la cámara.
- Los sensores que usan el modo de generación de informes continua, como acelerómetros y giroscopios, no reciben eventos.
- Los sensores que usan los modos de generación de informes ante un cambio o de acción única no reciben eventos.

Si tu aplicación debe detectar eventos de sensores en dispositivos con Android 9, deberás utilizar un servicio en primer plano.



Preguntas de repaso: *Sensores*

5.6. Introduciendo un misil en Asteroides

Para poder disparar a los asteroides será necesario introducir un misil en el juego. En el siguiente ejercicio aprenderemos a hacerlo.



Ejercicio: *Introduciendo un misil en Asteroides*

1. En primer lugar añade las siguientes variables a la clase `VistaJuego`:

```
// //// MISIL //////
private Grafico misil;
private static int PASO_VELOCIDAD_MISIL = 12;
private boolean misilActivo = false;
private int tiempoMisil;
```

2. Para trabajar con gráficos vectoriales, puedes crear en el constructor la variable `drawableMisil` de la siguiente forma:

```
ShapeDrawable dMisil = new ShapeDrawable(new RectShape());
dMisil.getPaint().setColor(Color.WHITE);
dMisil.getPaint().setStyle(Style.STROKE);
dMisil.setIntrinsicWidth(15);
dMisil.setIntrinsicHeight(3);
drawableMisil = dMisil;
```

3. Crea la variable `drawableMisil` para el caso de que se deseen gráficos en *bitmap*, utilizando el fichero `misil1.png`.
4. Inicializa el objeto `misil` de forma similar a como se ha hecho en `nave`.
5. En el método `onDraw()` dibuja `misil`, solo si lo indica la variable `misilActivo`.
6. Quita los comentarios de las llamadas a `activaMisil()`.
7. En el método `actualizaFisica()` añade las siguientes líneas:

```
// Actualizamos posición de misil
if (misilActivo) {
   misil.incrementaPos(factorMov);
   tiempoMisil-=factorMov;
   if (tiempoMisil < 0) {
      misilActivo = false;
   } else {
      for (int i = 0; i < asteroides.size(); i++)
      if (misil.verificaColision(asteroides.get(i))) {
         destruyeAsteroide(i);
         break;
      }
   }
}
```

8. Añade los siguientes dos métodos:

```
private void destruyeAsteroide(int i) {
   asteroides.remove(i);
   misilActivo = false;
   this.postInvalidate();
}
```

```
private void activaMisil() {
    misil.setCenX(nave.getCenX());
    misil.setCenY(nave.getCenY());
    misil.setAngulo(nave.getAngulo());
    misil.setIncX(Math.cos(Math.toRadians(misil.getAngulo())) *
                                         PASO_VELOCIDAD_MISIL);
    misil.setIncY(Math.sin(Math.toRadians(misil.getAngulo())) *
                                         PASO_VELOCIDAD_MISIL);
    tiempoMisil = (int) Math.min(this.getWidth() / Math.abs( misil.
          getIncX()), this.getHeight() / Math.abs(misil.getIncY())) - 2;
    misilActivo = true;
}
```

El primer método destruye el asteroide i y desactiva el misil. La llamada a `postInvalidate()` es necesaria para forzar el repintado de la vista. De no hacerse, el misil no desaparecería de la pantalla. Recuerda que este código se ejecuta en un hilo diferente al del interfaz de usuario, por lo que no podemos acceder a las vistas. Anteriormente hemos utilizado el método `invalidate()` para indicar al sistema que ha de redibujar una vista, este método solo ha de ser llamado desde el hilo principal. Para solicitar el redibujado una vista desde otros hilos has de utilizar `postInvalidate()`. Cuando el sistema regrese al hilo principal ya se encargará de realizarlo. El `this` no es imprescindible, se ha añadido para resaltar que pedimos la invalidación de nosotros mismos.

El segundo método permite activar un nuevo misil, este ha de partir del centro de la nave. El ángulo del misil ha de ser el mismo que el que tenga la nave. El módulo de la velocidad de la nave nos lo indica la constante `PASO_VELOCIDAD_MISIL`. Para descomponerla en sus componentes X e Y utilizamos el coseno y el seno. Recuerda que este juego tiene la peculiaridad de que lo que sale por un lado aparece por el otro. Por lo tanto, si disparáramos un misil, este podría acabar chocando contra la nave. Para solucionarlo vamos a dar un tiempo de vida al misil para impedir que pueda llegar de nuevo a la nave (`tiempoMisil`). Para obtener este tiempo obtenemos el mínimo entre la anchura dividida entre la velocidad en X y la altura dividida entre la velocidad en Y. Luego le restamos una constante. Terminamos activando el misil.

9. Verifica que todo funciona correctamente.

Ejercicio: *Introduciendo secciones críticas (synchronized) en Asteroides*

Cuando trabajamos con varios hilos es posible que se dé algún problema de acceso concurrente a los datos. En este código se accede al array `asteroides` desde el método `onDraw()` que se ejecuta en el hilo principal y desde el método `actualizaFisca()` que se ejecuta en el hilo secundario. Para evitar que los dos métodos se ejecuten de forma simultánea se propuso el ejercicio "Introduciendo secciones críticas en Java (synchronized)" al principio del capítulo. Consistía en añadir la palabra `synchronized` en la definición de ambos métodos. Las secciones críticas así definidas son demasiado grandes. Esto ocasiona que en algunos dispositivos con poca capacidad de proceso el sistema tarde algunos segundos en comenzar a dibujar los gráficos. Para resolverlo vamos a hacer las secciones críticas

lo más pequeñas posibles. Para ello has de tener en cuenta que los dos hilos pueden leer `asteroides` simultáneamente sin que se produzca ningún error. El problema aparece cuando un hilo está leyendo `asteroides`, el otro lo modifica.

En los métodos indicados elimina el código tachado e inserta el subrayado:

Hilo principal

```
synchronized void onDraw(Canvas canvas){
  ...
  synchronized (asteroides) {
   for (Grafico asteroide: asteroides){
     asteroide.dibujaGrafico(canvas);
   }
  }
}
```

Hilo secundario

```
synchronized void actualizaFisica(){
   ...
   for (Grafico asteroide: asteroides){
     asteroide.incrementaPos(factorMov);
   }
}

void destruyeAsteroide(int i) {
   synchronized (asteroides) {
       asteroides.remove(i);
       misilActivo = false;
   }
}
```

Analizando el código vemos como el problema no aparecerá cuando estando a mitad de ejecutar el bucle del método `onDraw()`, se cambie al hilo secundario y recorra el bucle para actualizar sus posiciones. Por el contrario, ocurría un error si estando a mitad de ejecutar el bucle del método `onDraw()`, se cambie al hilo secundario y se elimina un elemento de `asteroides`. Por esta razón se ha definido la sección crítica para que abarque solo este código.

Nota sobre Java: *La palabra clave* **synchronized** *permite definir una sección crítica asociada a un objeto. Este objeto va a actuar como una especie de identificador que hace posible tener varias secciones críticas. Todas los* **synchronized** *asociados a un mismo objeto realmente definen la misma sección crítica. Si se indica* **synchronized** *en un método, se asocia al objeto* this *de la clase actual. Cuando se utiliza* **synchronized** *dentro de un método resulta imprescindible indicar entre paréntesis el objeto que asociamos.*

En el código anterior podríamos haber utilizado **this** *para definir las secciones críticas. En este caso estaríamos asociando un objeto de tipo* `View`*, igual como se hacía antes de realizar los cambios. Se ha preferido asociarlo al objeto* **asteroides** *para así hacer hincapié en que los datos conflictivos por los que se define esta sección crítica están en este objeto. No obstante, ambas soluciones funcionan de forma correcta.*

1. Verifica que todo funciona correctamente.



Desafío: *Disparando varios misiles a la vez*

Tal y como se ha planteado el código, solo es posible lanzar un misil cada vez. Si disparamos un segundo misil, el primero desaparece. ¿Podrías modificar el código para que se pudieran lanzar tantos misiles cómo quisieras? Si no tienes muy claro por dónde empezar, a continuación se plantean los pasos para una posible solución:

1. Elimina la variable `misil` y en su lugar crea una lista de gráficos:

```
private ArrayList<Grafico> misiles;
```

2. Elimina la variable `misilActivo`. Cuando la lista de misiles esté vacía querrá decir que no hay ningún misil activo.

3. Elimina la variable `tiempoMisil` y en su lugar crea una lista de enteros:

```
private ArrayList<Integer> tiempoMisiles;
```

Los elementos de las *listas* `Misiles` y `tiempoMisiles` han de estar emparejados. Es decir, al misil en posición *x* de `misiles` le quedará un tiempo que se almacenará en la posición *x* de `tiempoMisiles`.



Nota sobre Java: *Observa como la variable* `tiempoMisil` *antes era de tipo* `int`*, pero ahora* `tiempoMisiles` *es una lista de* `Integer`*, no de* `int`*. Estos dos tipos de datos representan un número entero, pero el primero es una clase y el segundo un tipo simple. Hemos tenido que realizar este cambio dado que la clase* `ArrayList` *solo admite elementos que sean clases. Para más información, consúltese, en el anexo C, el apartado sobre envolventes (wrappers).*

4. En los métodos `onDraw()` y `actualizaFisica()` tendrás que añadir un bucle para recorrer todos los misiles.

5. La siguiente línea permite disminuir el elemento `m` de `tiempoMisil`:

```
tiempoMisiles.set(m, tiempoMisiles.get(m)-factorMov);
```


CAPÍTULO 6.
Multimedia y ciclo de vida de una actividad

Una de las funciones más habituales de los modernos teléfonos móviles es su utilización como reproductores multimedia. Su uso más frecuente es como reproductores MP3, para la reproducción de vídeos y televisión a través de Internet. La API de Android viene preparada con excelentes características de reproducción multimedia, permite la reproducción de una gran variedad de formatos, tanto de audio como de vídeo. El origen de los datos puede ser tanto un fichero local como un *stream* obtenido desde Internet. Todo este trabajo lo realiza principalmente la clase `MediaPlayer`.

Antes de comenzar la descripción de estos contenidos, comenzaremos el capítulo con un aspecto de vital importancia en el desarrollo de aplicaciones en Android: el ciclo de vida de una actividad. Es decir, cómo las actividades son creadas, ejecutadas, puestas en espera y finalmente destruidas.



Objetivos:

- Comprender el ciclo de vida de una actividad Android.
- Utilizar de forma correcta los diferentes eventos relacionados con el ciclo de vida.
- Aprender cuándo y cómo guardar el estado de una actividad.
- Repasar las facilidades multimedia disponibles en Android, qué formatos soporta y las clases que hemos de utilizar.
- Describir la clase MediaPlayer, utilizada para la reproducción de audio y vídeo.
- Dotar a la aplicación Asteroides de un control adecuado de su ciclo de vida y de varios efectos de audio.

6.1. Ciclo de vida de una actividad

El ciclo de vida de una aplicación Android es bastante diferente del **ciclo de vida de** una aplicación en otros SO, como Windows. La mayor diferencia es que, **en Android**, el ciclo de vida es controlado principalmente por el sistema, **en lugar de ser** controlado directamente por el usuario.


Vídeo[tutorial]: *Ciclo de vida de una aplicación en Android*

Una aplicación en Android está formada por un conjunto de elementos básicos de interacción con el usuario, conocidos como actividades. Además de varias actividades, una aplicación también puede contener servicios. El ciclo de vida de los servicios se estudiará en el capítulo 8. Son las actividades las que realmente controlan el ciclo de vida de las aplicaciones, dado que el usuario no cambia de aplicación, si no de actividad. El sistema mantiene una pila con las actividades previamente visualizadas, de forma que el usuario puede regresar a la actividad anterior pulsando la tecla "retorno".

Una aplicación Android corre dentro de su propio proceso Linux. Este proceso se crea con la aplicación y continuará vivo hasta que ya no sea requerido y el sistema reclame su memoria para asignársela a otra aplicación.

Una característica importante, y poco usual, de Android es que la destrucción de un proceso no es controlada directamente por la aplicación, si no que es el sistema el que determina cuándo destruir el proceso. Lo hace basándose en el conocimiento que tiene de las partes de la aplicación que están corriendo (actividades y servicios), en la importancia de dichas partes para el usuario y en cuánta memoria disponible hay en un determinado momento.

Si tras eliminar el proceso de una aplicación, el usuario vuelve a ella, se crea de nuevo el proceso, pero se habrá perdido el estado que tenía esa aplicación. En estos casos, será responsabilidad del programador almacenar el estado de las actividades, si queremos que cuando sean reiniciadas conserven su estado.

Como vemos, Android es sensible al ciclo de vida de una actividad; por lo tanto, necesitas comprender y manejar los eventos relacionados con el ciclo de vida si quieres crear aplicaciones estables.

Una actividad en Android puede estar en uno de estos cuatro estados:

Activa (*Running*): La actividad está encima de la pila, lo que quiere decir que es visible y tiene el foco.

Visible (*Paused*): La actividad es visible pero no tiene el foco. Se alcanza este estado cuando pasa a activa otra actividad con alguna parte transparente o que no ocupa toda la pantalla. Cuando una actividad está tapada por completo, pasa a estar parada.

Parada (*Stopped*): Cuando la actividad no es visible. El programador debe guardar el estado de la interfaz de usuario, las preferencias, etc.

Destruida (*Destroyed*): Cuando la actividad termina, al invocarse el método *finish()*, o cuando es matada por el sistema.

Cada vez que una actividad cambia de estado se generarán eventos que podrán ser capturados por ciertos métodos de la actividad. A continuación se muestra un esquema que ilustra los métodos que capturan estos eventos.

onCreate(Bundle): Se llama en la creación de la actividad. Se utiliza para realizar todo tipo de inicializaciones, como la creación de la interfaz de usuario o la inicialización de estructuras de datos. Puede recibir información de estado de la actividad (en una instancia de la clase `Bundle`), por si se reanuda desde una actividad que ha sido destruida y vuelta a crear.

onStart(): Nos indica que la actividad está a punto de ser mostrada al usuario.

onResume(): Se llama cuando la actividad va a comenzar a interactuar con el usuario. Es un buen lugar para lanzar las animaciones y la música.

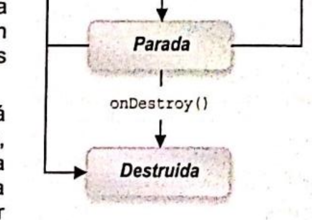
onPause(): Indica que la actividad está a punto de ser lanzada a segundo plano, normalmente porque se lanza otra actividad. Es el lugar adecuado para detener animaciones, música o almacenar los datos que estaban en edición.

Figura 4: *Ciclo de vida de una actividad.*

onStop(): La actividad ya no será visible para el usuario. ¡Ojo si hay muy poca memoria!: es posible que la actividad se destruya sin llamar a este método.

onRestart(): Indica que la actividad volverá a ser representada después de haber pasado por *onStop()*.

onDestroy(): Se llama antes de que la actividad sea totalmente destruida. Por ejemplo, cuando el usuario pulsa el botón de volver o cuando se llama al método *finish()*. ¡Ojo si hay muy poca memoria!: es posible que la actividad se destruya sin llamar a este método.


Ejercicio: *¿Cuándo se llama a los eventos del ciclo de vida?*

En este ejercicio vamos a implementar todos los métodos del ciclo de vida de la actividad principal y añadiremos un `Toast` para mostrar cuándo son ejecutados. De esta forma comprenderemos mejor cuándo se llama a cada método.

1. Abre la actividad `MainActivity` del proyecto Asteroides o **Mis Lugares**.
2. Añade en el método `onCreate()` el siguiente código:

```
Toast.makeText(this, "onCreate", Toast.LENGTH_SHORT).show();
```

3. Añade los siguientes métodos:

```
@Override protected void onStart() {
   super.onStart();
   Toast.makeText(this, "onStart", Toast.LENGTH_SHORT).show();
}

@Override protected void onResume() {
   super.onResume();
   Toast.makeText(this, "onResume", Toast.LENGTH_SHORT).show();
}

@Override protected void onPause() {
   Toast.makeText(this, "onPause", Toast.LENGTH_SHORT).show();
   super.onPause();
}

@Override protected void onStop() {
   Toast.makeText(this, "onStop", Toast.LENGTH_SHORT).show();
   super.onStop();
}

@Override protected void onRestart() {
   super.onRestart();
   Toast.makeText(this, "onRestart", Toast.LENGTH_SHORT).show();
}

@Override protected void onDestroy() {
   Toast.makeText(this, "onDestroy", Toast.LENGTH_SHORT).show();
   super.onDestroy();
}
```

```
override fun onStart() {
   super.onStart()
   Toast.makeText(this, "onStart", Toast.LENGTH_SHORT).show()
}

override fun onResume() {
   super.onResume()
   Toast.makeText(this, "onResume", Toast.LENGTH_SHORT).show()
}

override fun onPause() {
   Toast.makeText(this, "onPause", Toast.LENGTH_SHORT).show()
   super.onPause()
}

override fun onStop() {
   Toast.makeText(this, "onStop", Toast.LENGTH_SHORT).show()
   super.onStop()
}
```



```
override fun onRestart() {
   super.onRestart()
   Toast.makeText(this, "onRestart", Toast.LENGTH_SHORT).show()
}

override fun onDestroy() {
   Toast.makeText(this, "onDestroy", Toast.LENGTH_SHORT).show()
   super.onDestroy()
}
```

4. Ejecuta la aplicación y observa la secuencia de `Toast`.
5. Selecciona la opción *Acerca de...* y luego regresa a la actividad. Observa la secuencia de `Toast`.
6. Selecciona la opción *Preferencias* y luego regresa a la actividad. Observa la secuencia de `Toast`.
7. Sal de la actividad y observa la secuencia de `Toast`.



Ejercicio: *Aplicando eventos del ciclo de vida en la actividad Juego de Asteroides*

Asteroides gestiona el movimiento de los objetos gráficos por medio de un *thread* que se ejecuta continuamente. Cuando la aplicación pasa a segundo plano, este *thread* continúa ejecutándose, por lo que puede hacer que nuestro teléfono funcione más lentamente y, además, gaste más batería. Este problema aparece por una gestión incorrecta del ciclo de vida. En el siguiente ejercicio veremos cómo solucionarlo:

1. Abre el proyecto Asteroides y ejecútalo; preferiblemente en un terminal real.
2. Pulsa el botón *Jugar* y cuando esté en mitad de la partida pulsa el botón de inicio (o *Casa*) para dejar la actividad en estado *parada*. Las actividades en este estado no tendrían que consumir recursos. Sin embargo, Asteroides sí que lo hace. Para verificarlos utiliza el administrador de tareas (en las últimas versiones de Android, puedes abrirlo pulsando un segundo sobre el botón *Casa* y seleccionando el icono con forma de gráfico de tarta que aparece abajo a la izquierda). El resultado puede ser similar al que se muestra a la derecha:

NOTA: Si el administrador de tareas de tu terminal no te permite mostrar el porcentaje de uso de la CPU, te recomendamos que instales un programa que te lo permita.



Como puedes ver, la aplicación Asteroides está consumiendo casi el 50 % del uso de la CPU. Evidentemente, algo hemos hecho mal. No es lógico que, cuando la actividad `Juego` está en segundo plano, se siga llamando a `actualizaFisica()`. En este ejercicio aprenderemos a solucionarlo.

3. Incluye la siguiente variable en la actividad `Juego`:

```
private VistaJuego vistaJuego;
```

4. Al final de `onCreate` añade:

```
vistaJuego = findViewById(R.id.VistaJuego);
```

5. Incorpora los siguientes métodos a la actividad:

```
@Override protected void onPause() {
   vistaJuego.getThread().pausar();
   super.onPause();
}
@Override protected void onResume() {
   super.onResume();
   vistaJuego.getThread().reanudar();
}
@Override protected void onDestroy() {
   vistaJuego.getThread().detener();
   super.onDestroy();
}
```

Lo que intentamos hacer con este código es poner en pausa el *thread* secundario cuando la actividad deje de estar activa y reanudarlo cuando recupere el foco. Además, detener el *thread* cuando la actividad vaya a ser destruida.

NOTA: *Realmente, el thread será destruido al destruirse la actividad que lo ha lanzado. No obstante, puede resultar interesante hacerlo lo antes posible.*

Observa como los métodos llamados por eventos del ciclo de vida solo pueden escribirse en una `Activity`, por lo que no sería válido hacerlo en `VistaJuego`.

6. Abre la clase `VistaJuego`, busca la definición de la clase `ThreadJuego` y reemplázala por la siguiente:

```
class ThreadJuego extends Thread {
   private boolean pausa,corriendo;

   public synchronized void pausar() {
      pausa = true;
   }

   public synchronized void reanudar() {
      pausa = false;
      notify();
   }

   public void detener() {
      corriendo = false;
      if (pausa) reanudar();
   }

   @Override   public void run() {
      corriendo = true;
```



```
      while (corriendo) {
         actualizaFisica();
         synchronized (this) {
            while (pausa) {
               try {
                  wait();
               } catch (Exception e) {
               }
            }
         }
      }
   }
}
```

Comenzamos declarando las variables `pausa`, `corriendo`. Estas pueden ser modificadas mediante los métodos `pausar()`, `reanudar()` y `detener()`.

La palabra reservada `synchronized` impide el acceso de varios *threads* a una sección del código. En el capítulo anterior hemos explicado los hilos de ejecución. El método `run()` se ha modificado de manera que, en lugar de ser un bucle infinito, permitimos que termine poniendo la variable `corriendo` a `false`. Luego, tras llamar a `actualizaFisica()`, se comprueba si se ha activado `pausa`. En tal caso, se entra en un bucle donde ponemos en espera el *thread* llamando al método `wait()`. Este quedará bloqueado hasta que se llame a `notify()`. Esta acción se realizará desde el método `reanudar()`. La llamada a `wait()` puede lanzar excepciones, por lo que es obligatorio escribirla dentro de un bloque `try {…} catch {…}`.

7. Dado que la variable `thread` es de tipo `private`, no se puede manipular desde fuera de `VistaJuego`. Para poder llamar a los métodos de este objeto (pausar, reanudar, etc.) vamos a incluir un método *getter*. Para ello sitúa el cursor justo antes de la última llave de `VistaJuego`. Pulsa con el botón derecho en el código y selecciona la opción *Source > Generate Getters and Setters…* Marca solo el método `getThread()` tal y como se muestra a la derecha:

puntuacion
thread
getThread()
setThread(ThreadJuego)
ultimoProceso

Se insertará el siguiente código:

```
public ThreadJuego getThread() {
   return thread;
}
```

8. Ejecuta de nuevo la aplicación y repite el segundo punto de este ejercicio. En este caso el resultado ha de ser similar al siguiente:





Preguntas de repaso: *Ciclos de vida de una actividad*

6.1.1. ¿Qué proceso se elimina?

Como hemos comentado, Android mantiene en memoria todos los procesos que quepan aunque estos no se estén ejecutando. Una vez que la memoria está llena y el usuario decide ejecutar una nueva aplicación, el sistema ha de determinar qué proceso de los que están en ejecución ha de ser eliminado. Android ordena los procesos en una lista jerárquica, asignándole a cada uno de ellos una determinada "importancia". Esta lista se confecciona basándose en los componentes de la aplicación que están corriendo (actividades y servicios) y el estado de estos componentes.

Para establecer esta jerarquía de importancia se distinguen los siguientes tipos de procesos:

Proceso de primer plano (*Foreground process*): Hospeda una actividad en la superficie de la pantalla con la cual el usuario está interactuando (su método `onResume()` ha sido llamado). Debería haber solo uno o unos pocos procesos de este tipo. Solo serán eliminados cómo último recurso, si es que la memoria está tan baja que ni siquiera estos procesos pueden continuar corriendo.

Proceso visible (*Visible process*): Hospeda una actividad que está visible en la pantalla, pero no en el primer plano (su método `onPause()` ha sido llamado). Considerado importante, no será eliminado a menos que sea necesario para mantener los procesos de primer plano.

Proceso de servicio (*Service process*): Hospeda un servicio que ha sido inicializado con el método `startService()`. Aunque estos procesos no son directamente visibles para el usuario, generalmente están haciendo tareas que para él son importantes (tales como reproducir un archivo MP3 o mantener una conexión con un servidor de contenidos). El sistema siempre tratará de mantener esos procesos corriendo, a menos que los niveles de memoria comiencen a comprometer el funcionamiento de los procesos de primer plano o visibles.

Proceso de fondo (*Background process*): Hospeda una actividad que no es visible para el usuario (su método `onStop()` ha sido llamado). Si estos procesos son eliminados, no tendrán un impacto directo en la experiencia del usuario. Como hay muchos de estos procesos, el sistema debe asegurar que el último proceso visto por el usuario sea el último en ser eliminado.

Proceso vacío (*Empty process*): No hospeda ningún componente de aplicación activo. La única razón para mantener ese proceso es tener una caché que permita mejorar el tiempo de activación la próxima vez que un componente de su aplicación sea ejecutado.

Vídeo[tutorial]: *Ciclo de vida de los procesos en Android*



Vídeo[tutorial]: *El ciclo de vida de las aplicaciones en Android: Un ejemplo paso a paso*



Práctica: *Aplicando eventos del ciclo de vida en la actividad inicial*

Los conceptos referentes al ciclo de vida de una actividad son imprescindibles para el desarrollo de aplicaciones estables en Android. Para reforzar estos conceptos te proponemos el siguiente ejercicio, en el que vamos a reproducir una música de fondo en la actividad principal.

1. Abre el proyecto Asteroides o MisLugares.
2. Busca un fichero de audio (en este capítulo se listan los formatos soportados por Android). Renombra este fichero como *audio.xxx* y cópialo a la carpeta *res/raw.*

   NOTA: *Cada vez que ejecutes el proyecto, este fichero se añadirá al paquete .apk. Si este fichero es muy grande, la aplicación también lo será, lo que ralentizará su instalación. Para agilizar la ejecución te recomendamos un fichero muy pequeño; por ejemplo, un .mp3 de corta duración o un fichero MIDI (.mid). Si no encuentras ninguno, puedes descargar este:*

   http://www.androidcurso.com/images/dcomg/ficheros/audio.mid

3. Abre la actividad `MainActivity` y declara el siguiente objeto:

```
MediaPlayer mp;
```

4. Añade las siguientes líneas en el método `onCreate()`:

```
mp = MediaPlayer.create(this, R.raw.audio);
mp.start();
```

5. Ejecuta el proyecto y verifica que cuando sales de la actividad la música sigue sonando cierto tiempo.
6. Utilizando los eventos del ciclo de vida queremos que, cuando la actividad deje de estar **activa,** el audio deje de escucharse. Puedes utilizar los métodos:

```
mp.pause();
mp.start();
```

7. Verifica que funciona correctamente.



Práctica: *Aplicando eventos del ciclo de vida en la actividad inicial (II)*

1. Tras realizar el ejercicio anterior, ejecuta la aplicación y abre la actividad *Acerca de...* La música ha de detenerse.
2. Nos interesa que, mientras parte de esta actividad esté visible (como ha ocurrido en el punto anterior), la música se escuche. Es decir, utilizando los eventos del ciclo de vida queremos que, cuando la actividad deje de estar **visible,** el audio deje de escucharse.

3. Verifica que, cuando abres la actividad *Acerca de...*, la música continúa reproduciéndose.
4. Pasa ahora a una actividad que ocupe la totalidad de la pantalla (por ejemplo, la actividad `Juego` o `VistaLugarActivity`). En teoría, la música tendría que detenerse, dado que la actividad `MainActivity` ya no es visible y tendría que haber pasado a estado parada. Observa cómo la música acaba deteniéndose, pero es posible que tarde unos segundos. Esto se debe a que la llamada al método `onStop()` no es prioritaria, por lo que el sistema puede retardar su ejecución. En caso de tratarse de información visual, en lugar de acústica, este retardo no tendría una repercusión directa para el usuario, dado que la actividad no es visible.
5. Tras el problema detectado en el punto anterior, deshaz los cambios introducidos en esta práctica y deja la aplicación como se pedía en la práctica anterior.



Práctica: *Aplicando eventos del ciclo de vida en la actividad Juego para desactivar los sensores*

En la unidad anterior hemos aprendido a utilizar los sensores para manejar la nave en Asteroides. El uso de sensores ha de realizarse con mucho cuidado, dado su elevado consumo de batería. Resulta importante que, cuando nuestra actividad quede en un segundo plano, se detenga la lectura de los sensores, para que así no siga consumiendo batería.

1. Crea los métodos `activarSensores()` y `desactivarSensores()` en la clase `VistaJuego`.
2. Mueve la línea de código que activa los sensores (`mSensorManager.registerListener()`) al método `activarSensores()`.
3. Para desactivar los sensores hay que llamar al siguiente método:

   `mSensorManager.unregisterListener(SensorEventListener);`

4. El parámetro de este método corresponde al objeto `SensorEventListener` del que queremos que deje de recibir eventos. En nuestro caso, nosotros mismos (`this`). Utiliza este método dentro de `desactivarSensores()`.
5. Utiliza los métodos del ciclo de vida adecuados para activar o desactivar los sensores. Recuerda que estos métodos los recoge la actividad, por lo que tendrás que incluirlos en la clase `Juego`.

6.1.2. Guardando el estado de una actividad

Cuando el usuario ha estado utilizando una actividad y, tras cambiar a otras, regresa a la primera, lo habitual es que esta permanezca en memoria y continúe su ejecución sin alteraciones. Como hemos explicado, en situaciones de escasez de memoria, es posible que el sistema haya eliminado el proceso que ejecutaba la actividad. En ese caso, el proceso se creará de nuevo, pero se habrá perdido su estado, es decir, se habrá perdido el valor de sus variables. Como consecuencia, si el usuario estaba a mitad de un proceso de edición o estaba reproduciendo un audio en un punto determinado, perderá esa información. En este apartado estudiaremos un mecanismo sencillo que nos proporciona Android para resolver este problema.

NOTA: Cuando se ejecuta una actividad sensible a la inclinación del teléfono, es decir, puede verse en horizontal o en vertical, se presenta un problema similar al anterior. La actividad es destruida y vuelta a construir con las nuevas dimensiones de pantalla y, por lo tanto, se llama de nuevo al método `onCreate()`. Antes de que la actividad sea destruida también resulta fundamental guardar su estado.



Vídeo[tutorial]: *Guardar el estado de las actividades en Android*

Para guardar el estado de una actividad debes utilizar los siguientes dos métodos:

onSaveInstanceState(Bundle): Lo invoca el sistema cuando ha de destruir una actividad que más adelante ha de restaurar (por cambiar su inclinación o por falta de memoria), para permitir a la actividad guardar su estado.

onRestoreInstanceState(Bundle): Se invoca cuando se restaura la actividad para recuperar el estado guardado por `onSaveInstanceState()`.

NOTA: Nunca utilices el método `onSaveInstanceState()` para guardar los datos generados por una actividad (como guardar un formulario en una base de datos). Este método es llamado cuando el sistema debe destruir una actividad que va a ser recuperada en un futuro. Si esta actividad es destruida llamando a `finish()` o pulsando el botón atrás, el método `onSaveInstanceState()` no será llamado. Para guardar estos datos utiliza `onPause()` o `onStop()`.

Veamos un ejemplo de lo sencillo que resulta guardar la información de una variable tipo cadena de caracteres y entero.

```
String var;
int pos;

@Override protected void onSaveInstanceState(Bundle guardarEstado) {
   super.onSaveInstanceState(guardarEstado);
   guardarEstado.putString("variable", var);
   guardarEstado.putInt("posicion", pos);
}

@Override protected void onRestoreInstanceState(Bundle recEstado) {
   super.onRestoreInstanceState(recEstado);
   var = recEstado.getString("variable");
   pos = recEstado.getInt("posicion");
}
```

```
var v: String? = null
var pos: Int = 0

override fun onSaveInstanceState(guardarEstado: Bundle) {
   super.onSaveInstanceState(guardarEstado)
```

```
    guardarEstado.putString("variable", v)
    guardarEstado.putInt("posicion", pos)
}

override fun onRestoreInstanceState(recEstado: Bundle) {
    super.onRestoreInstanceState(recEstado)
    v = recEstado.getString("variable")
    pos = recEstado.getInt("posicion")
}
```



Práctica: *Guardando el estado en la actividad inicial*

1. Ejecuta el proyecto Asteroides o Mis Lugares.
2. Cambia de orientación el teléfono. Observarás como la música se reinicia cada vez que lo haces.
3. Utilizando los métodos para guardar el estado de una actividad, trata de que cuando se voltea el teléfono, el audio continúe en el mismo punto de reproducción. Puedes utilizar los siguientes métodos:

```
pos = mp.getCurrentPosition();
mp.seekTo(pos);
```

4. Verifica el resultado.



Solución: A continuación, se muestra una posible solución al ejercicio:

```
@Override protected void onSaveInstanceState(Bundle estadoGuardado){
    super.onSaveInstanceState(estadoGuardado);
    if (mp != null) {
        int pos = mp.getCurrentPosition();
        estadoGuardado.putInt("posicion", pos);
    }
}

@Override protected void onRestoreInstanceState(Bundle estadoGuardado){
    super.onRestoreInstanceState(estadoGuardado);
    if (estadoGuardado != null && mp != null) {
        int pos = estadoGuardado.getInt("posicion");
        mp.seekTo(pos);
    }
}
```

```
override fun onSaveInstanceState(estadoGuardado: Bundle) {
    super.onSaveInstanceState(estadoGuardado)
    if (mp != null) {
        val pos = mp.getCurrentPosition()
        estadoGuardado.putInt("posicion", pos)
```



```
    }
}
override fun onRestoreInstanceState(estadoGuardado: Bundle?) {
    super.onRestoreInstanceState(estadoGuardado)
    if (estadoGuardado != null && mp != null) {
        val pos = estadoGuardado.getInt("posicion")
        mp.seekTo(pos)
    }
}
```

Para reforzar el uso de estos métodos te recomendamos el ejercicio planteado en el apartado 6.4.1, donde se pide recordar el punto de reproducción de un vídeo.



Preguntas de repaso: *Guardar estado*

6.2. Utilizando multimedia en Android

La integración de contenido multimedia en nuestras aplicaciones resulta muy sencilla gracias a la gran variedad de facilidades que nos proporciona la API.

Concretamente, podemos reproducir audio y vídeo desde orígenes distintos:

- Desde un fichero almacenado en el dispositivo.
- Desde un recurso que está incrustado en el paquete de la aplicación (fichero `.apk`).
- Desde un *stream* que es leído desde una conexión de red. En este punto admite dos posibles protocolos (`http://` y `rstp://`).

También resulta sencilla la grabación de audio y vídeo, siempre que el *hardware* del dispositivo lo permita.

En la siguiente lista se muestran las clases de Android que nos permitirán acceder a los servicios multimedia:

MediaPlayer: Reproducción de audio/vídeo desde ficheros o *streams*.

MediaController: Visualiza controles estándar de un `mediaPlayer` (pausa, stop, etc.).

VideoView: Vista que permite la reproducción de vídeo.

MediaRecorder: Permite grabar audio y vídeo.

AsyncPlayer: Reproduce la lista de audios desde un *thread* secundario.

AudioManager: Gestiona varias propiedades del sistema (volumen, tonos...).

AudioTrack: Reproduce un búfer de audio PCM directamente por *hardware*.

SoundPool: Maneja y reproduce una colección de recursos de audio.

JetPlayer: Reproduce audio y vídeo interactivo creado con `JetCreator`.

Camera: Cómo utilizar la cámara para tomar fotos y vídeo.

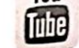
FaceDetector: Identifica la cara de la gente en un *bitmap*.

La plataforma Android soporta una gran variedad de formatos, muchos de los cuales pueden ser tanto decodificados como codificados. A continuación mostramos una tabla con los formatos multimedia soportados. No obstante, algunos modelos de móviles pueden soportar formatos adicionales que no se incluyen en la tabla, como por ejemplo DivX.

Cada desarrollador es libre de usar los formatos incluidos en el núcleo del sistema o aquellos que solo se incluyen en algunos dispositivos.

Tipo	Formato	Codifica	Decodifica	Detalles	Fichero soportado
Audio	AAC LC/LTP	X	X	Mono/estéreo con cualquier combinación estándar de frecuencia > 160 Kbps y ratios de muestreo de 8 a 48 kHz	3GPP (.3gp) MPEG-4 (.mp4) No soporta raw AAC (.aac) MPEG-TS (.ts)
	HE-AACv1	a partir 4.1	X		
	HE-AACv2		X		
	AAC ELD	a partir 4.1	a partir 4.1	Mono/estéreo, 16-48 kHz	
	AMR-NB	X	X	4.75 a 12.2 Kbps muestreada a @ 8 kHz	3GPP (.3gp)
	AMR-WB	X	X	9 ratios de 6.60 Kbps a 23.85 Kbps a @ 16 kHz	3GPP (.3gp)
	MP3		X	Mono/estéreo de 8 a 320 Kbps, tasa de bits constante (CBR) o variable (VBR)	MP3 (.mp3)
	MIDI		X	MIDI tipo 0 y 1. DLS v1 y v2. XMF y XMF móvil. Soporte para tonos de llamada RTTTL / RTX, OTA y iMelody	Tipo 0 y 1 (.mid, .xmf, .mxmf). RTTTL / RTX (.rtttl, .rtx), OTA (.ota) iMelody (.imy)
	Ogg Vorbis		X		Ogg (.ogg) Matroska (.mkv a partir 4.0)
	FLAC		a partir 3.1	Mono/estéreo (no multicanal)	FLAC (.flac)
	PCM/WAVE	a partir 4.1	X	8 y 16 bits PCM lineal (frecuencias limitadas por el *hardware*)	WAVE (.wav)
Imagen	JPEG	X	X	Base + progresivo	JPEG (.jpg)
	GIF		X		GIF (.gif)
	PNG	X	X		PNG (.png)
	BMP		X		BMP (.bmp)
	WebP	a partir 4.0	a partir 4.0		WebP (.webp)
Vídeo	H.263	X	X		3GPP (.3gp) MPEG-4 (.mp4)
	H.264 AVC	a partir 3.0	X	Baseline Profile (BP)	3GPP (.3gp) MPEG-4 (.mp4)
	MPEG-4 SP		X		3GPP (.3gp)
	VP8	a partir 4.3	a partir 2.3.3	*Streaming* a partir 4.0	WebM (.webm) Matroska (.mkv)

***Tabla 7:** Formatos multimedia soportados en Android.*



Vídeo[tutorial]: *Multimedia en Android*

6.3. La vista VideoView

La forma más fácil de incluir un vídeo en tu aplicación es incluir una vista de tipo `VideoView`. Veamos cómo hacerlo en el siguiente ejercicio.



Ejercicio: *Reproducir un vídeo con VideoView*

1. Crea una nueva aplicación.
2. Reemplaza el fichero `ref/layout/activity_main.xml` por:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <VideoView
        android:id="@+id/surface_view"
        android:layout_width="320px"
        android:layout_height="240px"/>
</LinearLayout>
```

3. Reemplaza el siguiente código en la clase `MainActivity`:

```
public class MainActivity extends Activity {
   private VideoView mVideoView;

   @Override public void onCreate(Bundle savedInstanceState){
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      mVideoView = findViewById(R.id.surface_view);
      //de forma alternative si queremos un streaming usar
      //mVideoView.setVideoURI(Uri.parse(URLstring));
      mVideoView.setVideoPath("/mnt/sdcard/video.mp4");
      mVideoView.start();
      mVideoView.requestFocus();
   }
}
```

En el parámetro del método `setVideoPath()` estamos indicando un fichero local.

4. Busca un fichero de vídeo en codificación MP4. Renombralo a *video.mp4*.
5. Es necesario almacenar un fichero de vídeo en el dispositivo. Para ello selecciona la pestaña *Device File Explorer* en la esquina inferior derecha

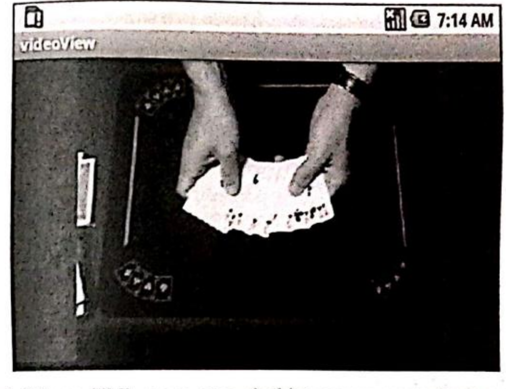
Device File Explorer
Emulator Nexus_4_API_25_Nuevo

Name	Permissions	Date	Size
mnt	drwxr-xr-x	2017-11-20 2	220 B
appfuse	drwx--x--x	2017-11-20 2	40 B
asec	drwxr-xr-x	2017-11-20 2	40 B
expand	drwxrwx--x	2017-11-20 2	40 B
media_rw	drwxr-x---	2017-11-20 2	60 B
obb	drwxr-xr-x	2017-11-20 2	40 B
runtime	drwx------	2017-11-20 2	100 B
sdcard	lrwxrwxrwx	2017-11-20 2	21 B
secure	drwx------	2017-11-20 2	60 B

6. Pulsa con el botón derecho sobre la carpeta *mnt/sdcard* y selecciona Upload... Selecciona el fichero *video.mp4*.
7. Ejecuta la aplicación. Observa que da un error de permisos.
8. Abre *AndroidManifest.xml* y añade el permiso `READ_EXTERNAL_STORAGE`.

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

9. Si utilizas una versión de Android 6 o superior, podrás comprobar que la aplicación se detiene. Este problema aparece dado que, a partir de la versión 6 de Android, el usuario ha de conceder los permisos considerados como peligrosos de forma explícita. Aprenderemos a hacer esto en el siguiente capítulo. De momento vamos a concederlos desde la configuración del dispositivo. Accede a Configuración / Aplicaciones / Nombre de la aplicación / Permisos y concede el permiso de "Almacen".
10. Ejecuta la aplicación y observa el resultado.



11. Modifica el fichero XML para que el vídeo aparezca centrado y ocupe toda la pantalla del teléfono, tal y como se muestra en la imagen de arriba
12. Añade la siguiente línea antes de la llamada al método `start()`:

```
mVideoView.setMediaController(new MediaController(this));
```

De esta forma permitimos que el usuario pueda controlar la reproducción del vídeo mediante el objeto `MediaController`.

13. Observa el resultado.

6.4. La clase MediaPlayer

La reproducción multimedia en Android se lleva a cabo principalmente por medio de la clase `MediaPlayer`. Veremos a continuación las características más importantes de esta clase y cómo podemos sacarle provecho.

Un objeto `MediaPlayer` puede pasar por una gran variedad de estados: inicializados sus recursos (*initialized*), preparando la reproducción (*preparing*), preparado para reproducir (*prepared*), reproduciendo (*started*), en pausa (*paused*), parado (*stopped*), reproducción completada (*playback completed*), finalizado (*end*) y con error (*error*). Resulta importante saber en qué estado se encuentra dado que muchos de los métodos solo pueden llamarse desde ciertos estados. Por ejemplo, no podremos ponerlo en reproducción (método `start()`) si no está en estado preparado. O no podremos ponerlo en pausa (`pause()`), si está parado. Si llamamos a un método no admitido para un determinado estado, se producirá un error de ejecución.

La siguiente tabla de transiciones permite conocer los métodos que podemos invocar desde cada uno de los estados y cuál es el nuevo estado al que iremos tras invocarlo. Existen dos tipos de métodos: los que no están subrayados representan métodos llamados de forma síncrona desde nuestro código, mientras que los que están subrayados representan métodos llamados de forma asíncrona por el sistema.

Estado salida	Estado entrada: Idle	Initialized	Preparing	Prepared	Started	Paused	Stopped	Playback Completed
Initialized	setDataSource							
Preparing		prepareAsync					prepareAsync	
Prepared		prepare	onPrepared	seekTo			prepare	
Started				start	seekTo start	start		start
Paused					pause	seekTo pause		
Stopped				stop	stop	stop	stop	stop
Playback Completed					onCompletion			seekTo
End	release	release	release	release	release	release	release	release
Error	onError	onError	onError	onError	onError	onError	onError	onError

Tabla 8: *Transiciones entre estados de la clase MediaPlayer.*



6.4.1. Reproducción de audio con MediaPlayer

Si el audio o vídeo se va a reproducir siempre en nuestra aplicación, resulta interesante incluirlo en el paquete `.apk` y tratarlo como un recurso. Este uso ya se ha ilustrado al comienzo del capítulo. Recordemos cómo se hacía:

1. Crea una nueva carpeta dentro de la carpeta `res` y llámala `raw`.
2. Arrastra a su interior el fichero de audio. Por ejemplo, `audio.mp3`.
3. Añade las siguientes líneas de código:

```
MediaPlayer mp = MediaPlayer.create(this, R.raw.audio);
mp.start();
```

Si deseas parar la reproducción, tendrás que utilizar el método `stop()`. Si a continuación quieres volver a reproducirlo, utiliza el método `prepare()` y luego `start()`. También puedes usar `pause()` y `start()` para ponerlo en pausa y reanudarlo.

Si en lugar de un recurso prefieres reproducirlo desde un fichero, utiliza el siguiente código. Observa como en este caso es necesario llamar al método `prepare()`. En el caso anterior no ha sido necesario dado que esta llamada se hace desde `create()`.

```
MediaPlayer mp = new MediaPlayer();
mp.setDataSource(RUTA_AL_FICHERO);
mp.prepare();
mp.start();
```

6.5. Un reproductor multimedia paso a paso

En el siguiente ejercicio vamos a profundizar en el objeto MediaPlayer por medio de un ejemplo, donde trataremos de realizar un reproductor de vídeos personalizado.



Ejercicio: *Un reproductor multimedia paso a paso*

1. Crea una nueva aplicación con los siguientes datos:

```
Template / Phone and Tablet / Empty Activity
Application name: VideoPlayer
Minimum SDK: API 19 Android 4.4 (KitKat)
```

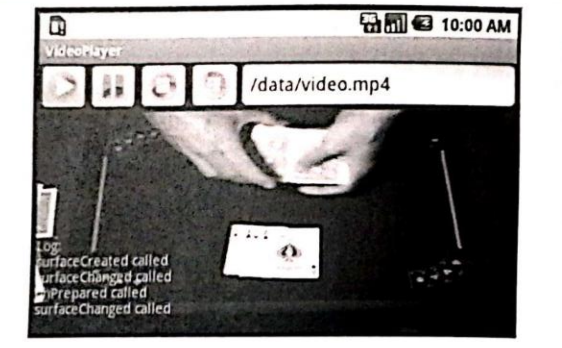
2. En la carpeta `res/drawable` arrastra los cuatro ficheros de iconos: `play.png`, `pause.png`, `stop.png` y `log.png`. Los puedes encontrar en http://www.androidcurso.com/images/dcomg/ficheros/Graficos.zip.

3. Reemplaza el fichero *res/layout/activity_main.xml* por:

```
<RelativeLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_height="match_parent"
   android:layout_width="match_parent">
    <LinearLayout
            android:id="@+id/ButonsLayout"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
            android:orientation="horizontal"
            android:layout_alignParentTop="true">
        <ImageButton android:id="@+id/play"
                     android:layout_height="wrap_content"
                     android:layout_width="wrap_content"
                     android:src="@drawable/play"/>
        <ImageButton android:id="@+id/pause"
                     android:layout_height="wrap_content"
                     android:layout_width="wrap_content"
                     android:src="@drawable/pause"/>
        <ImageButton android:id="@+id/stop"
                     android:layout_height="wrap_content"
                     android:layout_width="wrap_content"
                     android:src="@drawable/stop"/>
        <ImageButton android:id="@+id/logButton"
                     android:layout_height="wrap_content"
                     android:layout_width="wrap_content"
                     android:src="@drawable/log"/>
        <EditText    android:id="@+id/path"
                     android:layout_height="match_parent"
                     android:layout_width="match_parent"
                     android:text="/data/video.3gp"/>
    </LinearLayout>
    <VideoView  android:id="@+id/surfaceView"
                android:layout_height="match_parent"
                android:layout_width="match_parent"
                android:layout_below="@id/ButonsLayout"/>
    <ScrollView android:id="@+id/ScrollView"
                android:layout_height="100px"
                android:layout_width="match_parent"
```

```
                android:layout_alignParentBottom="true">
        <TextView  android:id="@+id/Log"
            android:layout_height="wrap_content"
            android:layout_width="match_parent"
                android:text="Log:"/>
    </ScrollView>
</RelativeLayout>
```

A continuación, se muestra la apariencia del *layout* anterior:



4. Reemplaza el código de la clase `MainActivity` por:

```
public class MainActivity extends AppCompatActivity implements
            OnBufferingUpdateListener, OnCompletionListener,
            MediaPlayer.OnPreparedListener, SurfaceHolder.Callback {
   private MediaPlayer mediaPlayer;
   private SurfaceView surfaceView;
   private SurfaceHolder surfaceHolder;
   private EditText editText;
   private ImageButton bPlay, bPause, bStop, bLog;
   private TextView logTextView;
   private boolean pause, stop;
   private String path;
   private int savePos = 0;

   public void onCreate(Bundle bundle) {
      super.onCreate(bundle);
      setContentView(R.layout.activity_main);
      surfaceView = findViewById(R.id.surfaceView);
      surfaceHolder = surfaceView.getHolder();
      surfaceHolder.addCallback(this);
      editText = findViewById(R.id.path);
      editText.setText("http://personales.gan.upv.es/~jtomas/video.3gp");
      logTextView = findViewById(R.id.Log);
      bPlay = findViewById(R.id.play);
      bPlay.setOnClickListener(new OnClickListener() {
         public void onClick(View view) {
            if (mediaPlayer != null) {
               if (pause) {
                  mediaPlayer.start();
               } else {
                  playVideo();
               }
            }
         }
      });
      bPause = findViewById(R.id.pause);
      bPause.setOnClickListener(new OnClickListener() {
         public void onClick(View view) {
            if (mediaPlayer != null && !stop) {
               pause = true;
               mediaPlayer.pause();
            }
         }
      });
      bStop = findViewById(R.id.stop);
      bStop.setOnClickListener(new OnClickListener() {
         public void onClick(View view) {
            if (mediaPlayer != null) {
               pause = false; stop = true;
               mediaPlayer.stop();
            }
         }
      });
      bLog = findViewById(R.id.LogButton);
      bLog.setOnClickListener(new OnClickListener() {
         public void onClick(View view) {
            if (logTextView.getVisibility()==TextView.VISIBLE) {
               logTextView.setVisibility(TextView.INVISIBLE);
            } else {
               logTextView.setVisibility(TextView.VISIBLE);
            }
         }
      });
      log("");
   }
```

Como puedes ver, la aplicación extiende la clase `Activity`. Además, implementamos cuatro interfaces que corresponden a varios escuchadores de eventos. Luego continúa la declaración de variables. Las primeras corresponden a diferentes elementos de la aplicación y su significado resulta obvio. Las variables `pause` y `stop` nos indican si el usuario ha pulsado el botón correspondiente, la variable `path` nos indica dónde está el vídeo en reproducción y la variable `savePos` almacena la posición de reproducción.

5. Añade:

```
private void playVideo() {
  try {
     pause = false; stop = false;
     path = editText.getText().toString();
     if (mediaPlayer!=null) mediaPlayer.release();
     mediaPlayer = new MediaPlayer();
```

```
      mediaPlayer.setDataSource(path);
      mediaPlayer.setDisplay(surfaceHolder);
      mediaPlayer.prepareAsync();
      mediaPlayer.setOnBufferingUpdateListener(this);
      mediaPlayer.setOnCompletionListener(this);
      mediaPlayer.setOnPreparedListener(this);
      mediaPlayer.setAudioStreamType(AudioManager.STREAM_MUSIC);
      mediaPlayer.seekTo(savePos);
   } catch (Exception e) {
      log("ERROR: " + e.getMessage());
   }
}
```

El código continúa con la definición del método `playVideo()`. Este método se encarga de obtener la ruta de reproducción y crear un nuevo objeto `MediaPlayer`, luego se le asigna la ruta y la superficie de visualización, a continuación se prepara la reproducción del vídeo. En caso de querer reproducir un *stream* desde la red, esta función puede tardar bastante tiempo, en cuyo caso es recomendable utilizar en su lugar el método `prepareAsync()`, que permite continuar con la ejecución del programa, aunque sin esperar a que el vídeo esté preparado. Las siguientes tres líneas asignan a nuestro objeto varios escuchadores de eventos que se describirán más adelante. Tras preparar el tipo de audio, se sitúa la posición de reproducción a los milisegundos indicados en la variable `savePos`. Si se trata de una nueva reproducción, esta variable será cero.

6. Añade el código:

```
public void onBufferingUpdate(MediaPlayer arg0, int percent) {
   log("onBufferingUpdate percent:" + percent);
}
public void onCompletion(MediaPlayer arg0) {
   log("onCompletion called");
}
```

Los métodos anteriores implementan las interfaces `OnBufferingUpdateListener` y `OnCompletionListener`. El primero irá mostrando el porcentaje de obtención de búfer de reproducción, mientras que el segundo será invocado cuando el vídeo en reproducción llegue al final.

7. Añade el código:

```
public void onPrepared(MediaPlayer mediaplayer) {
   log("onPrepared called");
   int mVideoWidth = mediaPlayer.getVideoWidth();
   int mVideoHeight = mediaPlayer.getVideoHeight();
   if (mVideoWidth != 0 && mVideoHeight != 0) {
      surfaceHolder.setFixedSize(mVideoWidth, mVideoHeight);
      mediaPlayer.start();
   }
}
```

El método anterior implementa la interfaz `OnPreparedListener`. Se invoca una vez que el vídeo ya está preparado para su reproducción. En ese momento podemos conocer su altura y anchura y ponerlo en reproducción.

8. Añade el código:

```
public void surfaceCreated(SurfaceHolder holder) {
   log("entramos en surfaceCreated ");
   playVideo();
}
public void surfaceChanged(SurfaceHolder surfaceholder,
                                             int i, int j, int k) {
   log("entramos en surfaceChanged");
}
public void surfaceDestroyed(SurfaceHolder surfaceholder) {
   log("entramos en surfaceDestroyed");
}
```

Los métodos anteriores implementan la interfaz `SurfaceHolder.Callback`. Se invocarán cuando nuestra superficie de visualización se cree, cambie o se destruya. Los métodos que siguen corresponden a acciones del ciclo de vida de una actividad.

9. Añade el código:

```
@Override protected void onDestroy() {
   super.onDestroy();
   if (mediaPlayer != null) {
      mediaPlayer.release();
      mediaPlayer = null;
   }
}
```

Este método se invoca cuando la actividad va a ser destruida. Dado que un objeto de la clase `MediaPlayer` consume muchos recursos, resulta interesante liberarlos lo antes posible.

10. Añade el código:

```
@Override public void onPause() {
   super.onPause();
   if (mediaPlayer != null & !pause) {
      mediaPlayer.pause();
   }
}

@Override public void onResume() {
   super.onResume();
   if (mediaPlayer != null & !pause) {
      mediaPlayer.start();
   }
}
```

Los dos métodos anteriores se invocan cuando la actividad pasa a un segundo plano y cuando vuelve a primer plano. Dado que queremos que el vídeo deje de reproducirse y continúe reproduciéndose en cada uno de estos casos, se invocan los métodos `pause()` y `start()`, respectivamente. No hay que confundir esta

acción con la variable pause, que indica es que el usuario ha pulsado el botón correspondiente.

11. Añade el código:

```
@Override protected void onSaveInstanceState(Bundle guardarEstado) {
   super.onSaveInstanceState(guardarEstado);

   if (mediaPlayer != null) {
      int pos = mediaPlayer.getCurrentPosition();
      guardarEstado.putString("ruta", path);
      guardarEstado.putInt("posicion", pos);
   }
}

@Override protected void onRestoreInstanceState(Bundle recEstado) {
   super.onRestoreInstanceState(recEstado);
   if (recEstado != null) {
      path = recEstado.getString("ruta");
      savePos = recEstado.getInt("posicion");
   }
}
```

Cuando este sistema necesita memoria, puede decidir eliminar nuestra actividad. Antes de hacerlo llamará al método onSaveInstanceState para darnos la oportunidad de guardar información sensible. Si más adelante el usuario vuelve a la aplicación, esta se volverá a cargar, invocándose el método onRestoreInstanceState, donde podremos restaurar el estado perdido. En nuestro caso, la información a guardar son las variables path y savePos, que representan el vídeo y la posición que estamos reproduciendo.

Ocurre el mismo proceso cuando el usuario cambia la posición del teléfono. Es decir, cuando el teléfono se voltea las actividades son destruidas y vueltas a crear, por lo que también se invocan estos métodos.

12. Añade el código:

```
private void log(String s) {
   logTextView.append("\n" + s);
}
} // fin de la clase
```

El último método es utilizado por varios escuchadores de eventos para mostrar información sobre lo que está pasando. Esta información puede visualizarse o no, utilizando el botón correspondiente.

13. Si quieres que tu reproductor pueda reproducir vídeos de la memoria SD debes añadir en *AndroidManifest.xml*:

```
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

14. Desde Android 9 el uso del protocolo HTTP no se recomienda, en su lugar hay que utilizar el protocolo HTTPS. Para permitir el tráfico HTTP en una aplicación Android has de añadir el siguiente atributo a la etiqueta <aplication> de AndroidManifest:


```
<application
    android:usesCleartextTraffic="true"
```

15. Ejecuta el proyecto y verifica su funcionamiento

16. Posiblemente el vídeo se ajuste ocupando todo el espacio disponible. Si prefieres que se conserve la relación de aspecto en el vídeo añade:

```
private int maxWidth = 0, maxHeight = 0;

@Override
public void onVideoSizeChanged(MediaPlayer mediaPlayer, int videoWidth,
int videoHeight) {
   if (maxWidth==0 && maxHeight==0) {
      maxWidth = surfaceView.getWidth();
      maxHeight = surfaceView.getHeight();
   }
   if (videoWidth>0 && videoHeight>0) {
      ViewGroup.LayoutParams videoParams = surfaceView.getLayoutParams();
      if (videoWidth / videoHeight > maxWidth / maxHeight) {
         videoParams.width = maxWidth;
         videoParams.height = maxWidth * videoHeight / videoWidth;
      } else {
         videoParams.height = maxHeight;
         videoParams.width = maxHeight * videoWidth / videoHeight;
      }
      surfaceView.setLayoutParams(videoParams);
   }
}
```

Este método será llamado cuando se conozca el tamaño del vídeo a reproducir y cada vez que este tamaño cambie. Vamos a intentar cambiar el tamaño de surfaceView para que mantenga la relación de aspecto del vídeo. Lo primero es obtener el tamaño máximo disponible. Como SurfaceView se ha añadido al layout ajustándose al contenedor, obtenemos su tamaño y lo guardamos en maxWidth y maxHeight. Pero esta operación solo se realiza la primera vez.

El cálculo principal se realiza en el siguiente if, solo si se ha indicado el tamaño del vídeo. Para cambiar los atributos de la vista usaremos una variable de tipo LayoutParams. Como queremos que el vídeo ocupe lo máximo posible, habrá que ajustarlo a la totalidad del ancho o a la totalidad del alto. El primer caso se dará cuando la relación de aspecto del vídeo sea mayor que la relación de aspecto de surfaceView. Una vez conocido el ancho o el alto de salida, no queda más que obtener la otra medida de forma que se mantenga la relación de aspecto.

17. Añade en el método playVideo() la línea:

```
mediaPlayer.setOnVideoSizeChangedListener(this);
```

18. En la definición de MainActivity haz que extienda de MediaPlayer.OnVideoSizeChangedListener.

6.6. Introduciendo efectos de audio con SoundPool

Hemos aprendido a utilizar la clase `MediaPlayer` para reproducir audio y vídeo. En un primer ejercicio vamos a aprender cómo introducir efectos de audio en Asteroides. Como veremos, esta clase no resulta adecuada para este uso. A continuación se introducirá la clase `SoundPool` y se mostrará mediante un ejercicio como esta sí que resulta eficiente para nuestro juego.



Ejercicio: *Introduciendo audio con MediaPlayer en Asteroides*

1. Abre el proyecto Asteroides.
2. Copia en la carpeta *res/raw* los ficheros: disparo.mp3 y explosion.mp3[30].
3. Abre la clase `VistaJuego` y añade las siguientes variables:

```
MediaPlayer mpDisparo, mpExplosion;
```

4. En el constructor de la clase inicialízalas de la siguiente forma:

```
mpDisparo = MediaPlayer.create(context, R.raw.disparo);
mpExplosion = MediaPlayer.create(context, R.raw.explosion);
```



5. Añade en el método `activaMisil()` de `VistaJuego`:

```
mpDisparo.start();
```

6. Añade en el método `destruyeAsteroide()` de `VistaJuego`:

```
mpExplosion.start();
```

7. Ejecuta el programa y verifica el resultado. ¿Qué pasa cuando disparamos o destruimos un asteroide? ¿El sonido se oye de forma inmediata?

La clase `SoundPool` maneja y reproduce de forma rápida recursos de audio en las aplicaciones. Un `SoundPool` es una colección de pistas de audio que se pueden cargar en la memoria desde un recurso (dentro de la APK) o desde el sistema de archivos. `SoundPool` utiliza el servicio de la clase `MediaPlayer` para descodificar el audio en un formato crudo (PCM de16 bits), lo que después permite reproducirlo rápidamente por el *hardware* sin tener que decodificarlo cada vez.

Los sonidos pueden repetirse en un bucle una cantidad establecida de veces, definiendo el valor al reproducirlo, o dejarse reproduciendo en un bucle infinito con -1. En este caso, será necesario detenerlo con el método `stop()`.

La velocidad de reproducción también se puede cambiar. El rango de velocidad de reproducción va de 0.5 a 2.0. Una tasa de reproducción de 1.0 hace que el sonido se reproduzca en su velocidad original. Una tasa de reproducción de 2.0 hace que el sonido se reproduzca al doble de su velocidad, y una tasa de reproducción de 0.5 hace que se reproduzca a la mitad de su velocidad.

---

[30] http://www.androidcurso.com/images/dcomg/ficheros/disparo.mp3, ...explosion.mp3

Cuando se crea un `SoundPool` hay que establecer en un parámetro del constructor el máximo de pistas que se pueden reproducir simultáneamente. Este parámetro no tiene por qué coincidir con el número de pistas cargadas. Cuando se pone una pista en reproducción (método `play()`) hay que indicar una prioridad. Esta prioridad se utiliza para decidir qué se hará cuando el número de reproducciones activas exceda el valor máximo establecido en el constructor. En este caso, se detendrá el flujo con la prioridad más baja, y en caso de que haya varios, se detendrá el más antiguo. En caso de que el nuevo flujo sea el de menor prioridad, este no se reproducirá. Puedes encontrar una lista de todos los métodos de esta clase en: http://developer.android.com/reference/android/media/SoundPool.html.



Ejercicio: *Introduciendo audio con SoundPool en Asteroides*

1. En el proyecto Asteroides elimina todo el código introducido a partir del punto 3 del ejercicio anterior.

2. Abre la clase `VistaJuego` y añade las siguientes variables:

```
// //// MULTIMEDIA //////
SoundPool soundPool;
int idDisparo, idExplosion;
```

3. En el constructor de la clase inicialízalas de la siguiente forma:

```
soundPool = new SoundPool( 5, AudioManager.STREAM_MUSIC , 0);
idDisparo = soundPool.load(context, R.raw.disparo, 0);
idExplosion = soundPool.load(context, R.raw.explosion, 0);
```

En el constructor de SoundPool hay que indicar tres parámetros: el primero corresponde al máximo de reproducciones simultáneas, el segundo es el tipo de *stream* de audio (normalmente, `STREAM_MUSIC`) y el tercero es la calidad de reproducción, aunque actualmente no se implementa.

Cada una de las pistas ha de ser cargada mediante el método `load()`. Existen cuatro sobrecargas para este método. La versión utilizada en el ejemplo permite cargar las pistas desde los recursos. El último parámetro corresponde a la prioridad, aunque actualmente no tiene ninguna utilidad.

4. Añade en el método `activaMisil()` de `VistaJuego`:

```
soundPool.play(idDisparo, 1, 1, 1, 0, 1);
```

El método `play()` permite reproducir una pista. Hay que indicarle: el identificador de pista, el volumen para el canal izquierdo y derecho (0.0 a 1.0), la prioridad, el número de repeticiones (-1 = siempre, 0 = solo una vez, 1 = repetir una vez, etc.) y la ratio de reproducción, con la que podremos modificar la velocidad o *pitch* (1.0 = reproducción normal; rango: 0.5 a 2.0).

5. Añade en el método `destruyeAsteroide()` de `VistaJuego`:

```
soundPool.play(idExplosion, 1, 1, 0, 0, 1);
```

Los parámetros utilizados para la explosión son similares, solo hemos introducido una prioridad menor. La consecuencia será que, si ya hay un total de 5 pistas reproduciéndose (véase constructor) y se pide la reproducción de un nuevo disparo, el sistema detendrá la reproducción de la explosión más antigua, por tener esta menos prioridad.

6. Ejecuta el programa y verifica el resultado. ¿Qué pasa cuando disparamos o destruimos un asteroide? ¿El sonido se oye de forma inmediata?

7. Modifica algunos de los parámetros correspondientes al método `play()` y verifica el resultado.

6.7. Grabación de audio

Las API de Android disponen de facilidades para capturar audio y vídeo, permitiendo su codificación en diferentes formatos. La clase `MediaRecorder` te permitirá de forma sencilla integrar esta funcionalidad en tu aplicación.

La mayoría de los dispositivos disponen de micrófono para capturar audio; sin embargo, esta facilidad no ha sido integrada en el emulador. Por lo tanto, has de probar los ejercicios de este apartado en un dispositivo real.

La clase `MediaRecorder` dispone de una serie de métodos que podrás utilizar para configurar y controlar la grabación:

`setAudioSource(int audio_source)` – Dispositivo que se utilizará como fuente del sonido. Normalmente, `MediaRecorder.AudioSource.MIC`. También se pueden utilizar otras constantes, como `DEFAULT`, `CAMCORDER`, `VOICE_CALL`, `VOICE_COMMUNICATION`, etc.

`setOutputFile (String fichero)` – Nombre del fichero de salida.

`setOutputFormat(int output_format)` – Formato del fichero de salida. Se pueden utilizar constantes de la clase `MediaRecorder.OutputFormat`: `DEFAULT`, `AMR_NB`, `AMR_WB`, `RAW_AMR` (ARM), `MPEG_4` (MP4) y `THREE_GPP` (3GPP).

`setAudioEncoder(int audio_encoder)` – Codificación del audio. Cuatro posibles constantes de la clase `MediaRecorder.AudioEncoder`: `AAC`, `AMR_NB`, `AMR_WB` y `DEFAULT`.

`setAudioChannels(int numeroCanales)` – Especificamos el número de canales 1: mono y 2: estéreo.

`setAudioEncodingBitRate (int bitRate)` (desde el nivel de API 8) – Especificamos los bits por segundo (bps) a utilizar en la codificación.

`setAudioSamplingRate (int samplingRate)` (desde el nivel de API 8) – Especificamos el número de muetras por segundo a utilizar en la codificación.

`setMaxDuration (int max_duration_ms)` (desde el nivel de API 3) – Indicamos una duración máxima para la grabación. Tras ese tiempo se detendrá.



`setMaxFileSize (long max_filesize_bytes)` (desde el nivel de API 3) – Indicamos un tamaño máximo para el fichero. Al alcanzar el tamaño se detendrá.

`prepare()` – Prepara la grabación para la captura de datos. Antes de llamarlo hay que configurar la grabación y después ya podemos invocar el método `start()`.

`start()` – Comienza la captura.

`stop()` – Finaliza la captura.

`reset()` – Reinicia el objeto como si lo acabáramos de crear. Hay que volver a configurarlo.

`release()` – Libera todos los recursos utilizados de forma inmediata. Si no llamas al método, se liberarán cuando el objeto sea destruido.

La clase `MediaRecorder` también dispone de métodos que podrás utilizar para configurar la grabación de vídeo. Más información en: http://developer.android.com/reference/android/media/MediaRecorder.html.

La siguiente tabla de transiciones muestra los diferentes métodos que pueden ser ejecutados en cada estado y el estado que alcanzaremos tras la llamada:

Estado salida	Estado entrada: Initial	Initialized	DataSource Configured	Prepared	Recording	Error
Initial		reset	reset	reset	reset stop	reset
Initialized	setAudioSource setVideoSource	setAudioSource setVideoSource				
DataSource Configured		setOutputFormat	setAudioEncoder setVideoEncoder setOutputFile setVideoSize setVideoFrameRate setPreviewDisplay			
Prepared			prepare			
Recording				start		
Released	releas					
Error	llamada incorrecta	llamada incorrecta	llamada incorrecta	llamada incorrecta	llamada incorrecta	

Tabla 9: *Transiciones entre estados de la clase MediaRecorder.*

Ejercicio: *Grabación de audio utilizando MediaRecorder*

1. Crea un nuevo proyecto con nombre *Grabadora* y tipo *Empty Activity*.
2. Reemplaza el *layout activity_main.xml* por el siguiente código:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <Button android:id="@+id/bGrabar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Grabar"
            android:onClick="grabar"/>
    <Button android:id="@+id/bDetener"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Detener Grabacion"
            android:onClick="detenerGrabacion"/>
    <Button android:id="@+id/bReproducir"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Reproducir"
            android:onClick="reproducir"/>
</LinearLayout>
```

3. Reemplaza el código de la actividad por:

```
public class MainActivity extends AppCompatActivity {
   private static final String LOG_TAG = "Grabadora";
   private MediaRecorder mediaRecorder;
   private MediaPlayer mediaPlayer;
   private static String fichero = Environment.
         getExternalStorageDirectory().getAbsolutePath()+"/audio.3gp";
   @Override public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
   }
   public void grabar(View view) {
      mediaRecorder = new MediaRecorder();
      mediaRecorder.setOutputFile(fichero);
      mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
      mediaRecorder.setOutputFormat(MediaRecorder.
                                 OutputFormat.THREE_GPP);
      mediaRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.
                                                   AMR_NB);
      try {
         mediaRecorder.prepare();
      } catch (IOException e) {
         Log.e(LOG_TAG, "Fallo en grabación");
      }
      mediaRecorder.start();
   }
   public void detenerGrabacion(View view) {
      mediaRecorder.stop();
      mediaRecorder.release();
   }
   public void reproducir(View view) {
      mediaPlayer = new MediaPlayer();
```



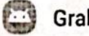
```
      try {
         mediaPlayer.setDataSource(fichero);
         mediaPlayer.prepare();
         mediaPlayer.start();
      } catch (IOException e) {
         Log.e(LOG_TAG, "Fallo en reproducción");
      }
   }
}
```

Comenzamos declarando dos objetos de las clases `MediaRecorder` y `MediaPlayer`, que serán usados para la grabación y la reproducción, respectivamente. También se declaran dos `String`: la constante `LOG_TAG` será utilizada como etiqueta para identificar nuestra aplicación en el fichero de Log; la variable `fichero` identifica el nombre del fichero donde se guardará la grabación. Este fichero se almacenará en una carpeta especialmente creada para nuestra aplicación (en el capítulo 9 se introducirá la gestión de ficheros). En el método `onCreate()` no se realiza ninguna acción especial. A continuación se han introducido tres métodos que se ejecutarán al pulsar los botones de nuestro *layout*. El significado de cada uno de los métodos que se invocan acaba de ser explicado.

4. Abre *AndroidManifest.xml* y añade los permisos `RECORD_AUDIO` y `WRITE_EXTERNAL_STORAGE`.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

5. Ejecuta la aplicación y trata de realizar una grabación.
6. Si utilizas una versión de Android 6 o superior, podrás comprobar que la aplicación se detiene. Si observas el LogCat, no encontrarás muchas pistas sobre el problema. Este aparece dado que a partir de la versión 6 de Android, el usuario ha de conceder los permisos considerados como peligrosos de forma explícita. Aprenderemos a hacer esto en el siguiente capítulo. De momento lo vamos a concederlos desde la configuración del dispositivo. Accede a Configuración / Aplicaciones / Grabadora / Permisos y concede los dos permisos que se muestran:



7. Ejecuta de nuevo la aplicación y verifica el resultado.



Ejercicio: *Grabación de audio utilizando MediaRecorder (II)*

La aplicación anterior resulta algo confusa de utilizar. Sería más sencillo si los botones que no pudiéramos utilizar en un determinado momento estuvieran deshabilitados. Para ello vamos a utilizar el método `Button.setEnabled(boolean)`:

1. Declara las siguientes tres variables al principio de la actividad:

```
private Button bGrabar, bDetener, bReproducir;
```

2. En el método `onCreate()` inicializa estos tres botones. Además, comenzaremos deshabilitando dos de los botones que al principio de la aplicación no deben ser pulsados:

```
bGrabar = findViewById(R.id.bGrabar);
bDetener = findViewById(R.id.bDetener);
bReproducir = findViewById(R.id.bReproducir);
bDetener.setEnabled(false);
bReproducir.setEnabled(false);
```

3. En el método `grabar()` añade el siguiente código:

```
bGrabar.setEnabled(false);
bDetener.setEnabled(true);
bReproducir.setEnabled(false);
```

4. En el método `detenerGrabacion()` añade el siguiente código:

```
bGrabar.setEnabled(true);
bDetener.setEnabled(false);
bReproducir.setEnabled(true);
```

5. Ejecuta de nuevo la aplicación y verifica el resultado.



Práctica: *Mejorando preferencias en Asteroides*

1. Modifica las preferencias de usuario para que se pueda configurar la reproducción de música de fondo y los efectos de audio.



Preguntas de repaso: *Multimedia*

CAPÍTULO 7. Seguridad y posicionamiento

En este capítulo abordaremos dos de los aspectos más novedosos de Android: la seguridad y la API de posicionamiento.

El capítulo comienza estudiando los fundamentos del sistema de seguridad que incorpora Android. Se trata de un aspecto vital para protegernos de aplicaciones mal intencionadas que intenten violar la privacidad del usuario y evitar que realicen acciones no deseadas. Gracias al sistema de permisos, se consigue impedir que las aplicaciones realicen acciones comprometidas, si previamente no han solicitado el permiso adecuado.

En la segunda parte del capítulo, se describe la API que incorpora Android para permitir conocer la posición geográfica del dispositivo. Estos servicios se basan principalmente en el GPS, pero también disponemos de novedosos servicios de localización basados en telefonía móvil y redes Wi-Fi. A lo largo de este capítulo mostraremos una serie de ejemplos que te permitirán aprender a utilizar estas funciones.

Terminamos el capítulo describiendo cómo podemos incorporar a nuestra aplicación servicios realizados por terceros. En concreto instalaremos una vista que permite representar un mapa de Google Maps.



Objetivos:

- Mostrar los pilares de la seguridad en Android.
- Describir cómo crea Android un usuario Linux asociado a cada aplicación.
- Describir el esquema de permisos en Android y enumerar los permisos más importantes.
- Mostrar cómo pueden ser ampliados los permisos de Android con permisos definidos por el usuario y enumerar los pasos a seguir para crear un nuevo permiso.

- Describir las API de Android para la geolocalización y los diferentes tipos de sistemas de posicionamiento disponibles.
- Ver lo sencillo que resulta incorporar en nuestra aplicación un servicio de un tercero. En concreto, Google Maps.

7.1. Los tres pilares de la seguridad en Android



Vídeo[tutorial]: *Seguridad en Android*

La seguridad es un aspecto clave de todo sistema. Si nos descargáramos una aplicación maliciosa de Internet o de Google Play Store, esta podría leer nuestra lista de contactos, averiguar nuestra posición GPS, mandar toda esta información por Internet y terminar enviando 50 mensajes SMS.

En algunas plataformas antiguas, como Windows Mobile, estábamos prácticamente desprotegidos ante aplicaciones maliciosas. Por lo tanto, los usuarios tenían que ser muy cautos antes de instalar una aplicación.

En otras plataformas, como en iOS, toda aplicación ha de ser validada por Apple antes de poder ser instala en un terminal. Además, solo está permitido instalar aplicaciones de la tienda oficial de Apple. Esto limita a los pequeños programadores y da un poder excesivo a Apple. Se trata de un planteamiento totalmente contrario al *software* libre.

Android propone un esquema de seguridad que protege a los usuarios, sin la necesidad de imponer un sistema centralizado y controlado por una única empresa. La seguridad en Android se fundamenta en los tres pilares siguientes:

- Como se ha comentado en el primer capítulo, Android está basado en Linux; por lo tanto, vamos a poder aprovechar la seguridad que incorpora este sistema operativo. De esta forma, Android puede impedir que las aplicaciones tengan acceso directo al *hardware* o interfieran con recursos de otras aplicaciones.
- Toda aplicación ha de ser firmada con un certificado digital que identifique a su autor. La firma digital también nos garantiza que el fichero de la aplicación no ha sido modificado. Si se desea modificar la aplicación, esta tendrá que ser firmada de nuevo, cosa que solo podrá hacer el propietario de la clave privada.
- Si queremos que una aplicación tenga acceso a partes del sistema que pueden comprometer la seguridad del sistema, hemos de utilizar un modelo de permisos, de forma que el usuario conozca los riesgos antes de instalar la aplicación.



Preguntas de repaso: *Los tres pilares de la seguridad*

7.1.1. Ejecución en procesos independientes Linux

Cada aplicación Android va a ser ejecutada en un proceso Linux independiente. Esto va a limitar su acceso directo al hardware y que pueda interferir con otras aplicaciones. Es lo que se conoce como ejecución en caja de arena.

Para impedir que otras aplicaciones puedan acceder a los ficheros creados por nuestra aplicación, Android crea una cuenta de usuario Linux (*user ID*) nueva por cada paquete (*.apk*) instalado en el sistema. Este usuario se crea cuando se instala la aplicación y permanece hasta que la aplicación es desinstalada.

Cualquier dato almacenado por la aplicación será asignado a su usuario Linux, por lo que normalmente no tendrán acceso otras aplicaciones. No obstante, cuando crees un fichero puedes usar los modos `MODE_WORLD_READABLE` y/o `MODE_WORLD_WRITEABLE` para permitir que otras aplicaciones puedan leer o escribir en el fichero. Aunque otras aplicaciones puedan escribir el fichero, el propietario siempre será el usuario asignado a la aplicación que lo creó.

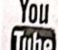
Dado que las restricciones de seguridad se garantizan a nivel de proceso, el código de dos paquetes no puede, normalmente, ejecutarse en el mismo proceso. Para ello sería necesario usar el mismo usuario. Puedes utilizar el atributo `sharedUserId` en `AndroidManifest.xml` para asignar un mismo usuario Linux a dos aplicaciones. Con esto conseguimos que, a efectos de seguridad, ambas aplicaciones sean tratadas como una sola. Por razones de seguridad, ambas aplicaciones han de estar firmadas con el mismo certificado digital.



Preguntas de repaso: *Usuario Linux*

7.1.2. Firma digital de los apks

Cuando publicamos un fichero apk con nuestra aplicación, este ha de estar firmado digitalmente. La firma digital es un sistema criptográfico que garantiza quien ha generado un documento electrónico y, además, que este documento no ha sido alterado. Un documento firmado digitalmente puede tener la misma validez jurídica que un documento firmado de puño y letra. Esto posibilita que gran número de trámites puedan realizarse a través de la red. Recomendamos ver el siguiente vídeo para conocer cómo se utiliza la firma digital:



Vídeo[tutorial]: *La firma digital*

Es habitual que un certificado digital sea validado por una autoridad de certificación. Sin embargo, en Android no se utiliza la figura de autoridad de certificación. Es el mismo desarrollador quien crea el certificado digital con el que firma el apk. La clave privada ha de guardarla a buen recaudo y la clave pública la adjuntará en el mismo apk. Cuando el usuario va a instalar el apk, extrae la clave pública y verifica que puede desencriptar el contenido.

Al no existir la autoridad de certificación cuando creamos el certificado los datos de desarrollador (nombre de la empresa, dirección...) pueden ser inventados. Esto parece contradictorio. ¿Para qué crear firmar con un certificado digital donde no se garantiza los datos del desarrollador? No obstante, esta firma digital sí que va a tener una gran cantidad de funciones:

- Podemos garantizar que, si el certificado es el original, el apk no ha podido ser modificado. Si alguien quiere modificar la aplicación ha de desencriptarla, modificarla y luego volverla a firmar. Pero como no disponen de la clave privada tendrán que firmar con un certificado diferente. Esto ocurre en tiendas de apps como Aptoide. Podrás encontrar muchas apps que han sido modificadas para eliminar anuncios o la necesidad de pagar. Todas estas aplicaciones son firmadas por un certificado digital diferente al original.
- En tiendas de apps, donde se garantiza la autoría del desarrollador (como Google Play), un desarrollador ha de subir una aplicación usando su propio certificado. Cuando se quiere subir una nueva versión, Google nos exige que usemos el mismo certificado digital. Esto garantiza que solo el que disponga de este certificado, puede subir actualizaciones.
- Cuando utilizamos una API de terceros, como Google Maps o Firebase, se nos va a pedir una huella digital del certificado de la aplicación. Esta es la forma habitual de identificar a nuestra aplicación. Al ser algo que no puede modificarse, la empresa que nos da el servicio podrá registrar el uso que hacemos, para luego facturarnos.
- Como hemos descrito en el punto anterior, si queremos que dos aplicaciones se ejecuten en un mismo proceso o compartan los mismos ficheros, es obligatorio que ambas estén firmadas por el mismo certificado. De esta forma se garantiza que han sido desarrolladas por el mismo programador.



Preguntas de repaso: *Firma digital de las aplicaciones*

7.1.3. El esquema de permisos en Android

Para proteger ciertos recursos y características especiales, Android define un esquema de permisos. Toda aplicación que acceda a estos recursos está obligada a declarar su intención de usarlos. En caso de que una aplicación intente acceder a un recurso del que no ha solicitado permiso, se generará una excepción de permiso y la aplicación será interrumpida inmediatamente.



Vídeo[tutorial]: *Los permisos en Android*

Cuando el usuario instala una aplicación, podrá examinar la lista de permisos que solicita la aplicación y decidir si considera oportuno instalar dicha aplicación. A partir de la versión 6, Android clasifica los permisos en peligrosos y normales. Como veremos en el siguiente apartado, a partir de esta versión, el usuario va a poder conceder o retirar los permisos peligrosos en cualquier momento. A continuación se muestra una lista con todos los permisos que pueden solicitar nuestras aplicaciones.

PERMISOS PELIGROSOS:

 Almacenamiento Externo:

- WRITE_EXTERNAL_STORAGE – Modificar/eliminar almacenamiento USB (API 4). Permite el borrado y la modificación de archivos en la memoria externa. Lo ha de solicitar toda aplicación que necesite escribir un fichero en la memoria externa; por ejemplo, exportar datos en XML. Pero al permitirlo también podrán modificar/eliminar ficheros externos creados por otras aplicaciones.
- READ_EXTERNAL_STORAGE – Leer almacenamiento USB (API 16) Permite leer archivos en la memoria externa. Este permiso se introdujo en la versión 4.1. En versiones anteriores todas las aplicaciones pueden leer en la memoria externa. Por lo tanto, has de tener cuidado con la información que dejas en ella.

Ubicación:

- ACCESS_COARSE_LOCATION – Localización no detallada (basada en red). Localización basada en telefonía móvil (*Cell-ID*) y Wi-Fi. Aunque en la actualidad esta tecnología suele ofrecernos menos precisión que el GPS, no siempre es así. Por ejemplo, se está aplicando en el interior de aeropuertos y museos con precisiones similares.
- ACCESS_FINE_LOCATION – Localización GPS detallada. Localización basada en satélites GPS. Al dar este permiso también estamos permitiendo la localización basada en telefonía móvil y Wi-Fi (ACCESS_COARSE_LOCATION).

Teléfono:

- CALL_PHONE – Llamar a números de teléfono directamente Servicios por los que tienes que pagar. Permite realizar llamadas sin la intervención del usuario. Nunca solicites este permiso en tus aplicaciones, muchos usuarios no instalarán tu aplicación. Si has de realizar una llamada, es mejor realizarla por medio de una intención. A diferencia de la llamada directa, no necesitas ningún permiso, dado que el usuario ha de pulsar el botón de llamada para que comience.
- READ_PHONE_STATE – Consultar identidad y estado del teléfono. Muchas aplicaciones, como los juegos, piden este permiso para ponerse en pausa cuando recibes una llamada. Sin embargo, también permite el acceso al número de teléfono, IMEI (identificador de teléfono GSM), IMSI (identificador de tarjeta SIM) y al identificador único de 64 bits que Google asigna a cada terminal. Incluso si hay una llamada activa, podemos conocer el número al que se conecta la llamada.

- READ_PHONE_NUMBERS – Leer los números almacenados en el dispositivo. Está incluido en las capacidades del permiso READ_PHONE_STATE pero se permite su utilización independiente en las Instant Apps.
- READ_CALL_LOG y WRITE_CALL_LOG – Leer y modificar el registro de llamadas telefónicas. Como realizar estas acciones se describe al final del capítulo 9.
- ADD_VOICEMAIL – Añadir mensajes de voz. Permite crear nuevos mensajes de voz en el sistema.
- USE_SIP – Usar Session Initial Protocol (API 9). Permite a tu aplicación usar el protocolo SIP.
- PROCESS_OUTGOING_CALLS – Procesar llamadas salientes. Permite a la aplicación controlar, modificar o abortar las llamadas salientes.
- ANSWER_PHONE_CALLS – Contestar llamadas entrantes.


Mensajes de texto (SMS):

- SEND_SMS – Enviar mensaje SMS Servicios por los que tienes que pagar. Permite a la aplicación mandar de texto SMS sin la validación del usuario. Por iguales razones que CALL_PHONE, a no ser que tu aplicación tenga que mandar SMS sin la intervención del usuario, resulta más conveniente enviarlos por medio de una intención.
- RECEIVE_SMS – Recibir mensajes de texto. Permite a la aplicación recibir y procesar SMS. Una aplicación puede modificar o borrar los mensajes recibidos.
- READ_SMS – Leer mensajes de texto. Permite a la aplicación leer los mensajes SMS entrantes.
- RECEIVE_MMS – Recibir mensajes MMS. Permite monitorizar los mensajes multimedia entrantes, pudiendo acceder a su contenido.
- RECEIVE_WAP_PUSH – Recibir mensajes WAP Push. Permite monitorizar los mensajes WAP Push entrantes. Un mensaje WAP PUSH es un tipo de SMS que se usa para acceder de manera sencilla a una página WAP en lugar de teclear su dirección URL en el navegador.

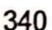

Contactos:

- READ_CONTACTS – Leer datos de contactos. Permiten leer información sobre los contactos almacenados (nombres, correos electrónicos, números de teléfono). Algunas aplicaciones podrían utilizar esta información de forma no lícita.
- WRITE_CONTACTS – Escribir datos de contactos. Permiten modificar los contactos.
- GET_ACCOUNTS – Obtener Cuentas. Permiten acceder a la lista de cuentas en el Servicio de Cuentas[31].


Calendario:

- READ_CALENDAR – Leer datos de calendario. Permite leer información del calendario del usuario.

[31] http://developer.android.com/intl/es/reference/android/accounts/AccountManager.html



- WRITE_CALENDAR – Escribir datos de calendario. Permite escribir en el calendario, pero no leerlo.


Cámara:

- CAMERA – Hacer fotos / grabar vídeos. Permite acceso al control de la cámara y a la toma de imágenes y vídeos. El usuario puede no ser consciente.

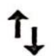

Micrófono:

- RECORD_AUDIO – Grabar audio. Permite grabar sonido desde el micrófono del teléfono.


Sensores corporales:

- BODY_SENSORS – Leer sensores corporales. Da acceso a los datos de los sensores que están monitorizando el cuerpo del usuario. Por ejemplo, el lector de ritmo cardiaco.

PERMISOS NORMALES:


Comunicaciones:

- INTERNET – Acceso a Internet sin límites. Permite establecer conexiones a través de Internet. Este es un permiso muy importante, en el que hay que fijarse a quién se otorga. La mayoría de las aplicaciones lo piden, pero no todas lo necesitan. Cualquier *malware* necesita una conexión para poder enviar datos de nuestro dispositivo.
- ACCESS_NETWORK_STATE – Ver estado de red. Información sobre todas las redes. Por ejemplo para saber si tenemos conexión a Internet.
- CHANGE_NETWORK_STATE – Cambiar estado de red. Permite cambiar el estado de conectividad de redes.
- NFC – Near field communication (API 9). Permite realizar operaciones de entrada/salida a través de NFC.
- TRANSMIT_IR – Transmitir por infrarrojos (API 19). Algunos dispositivos disponen de un trasmisor infrarrojo para el control remoto de electrodomésticos.


Conexión WiFi:

- ACCESS_WIFI_STATE – Ver estado de Wi-Fi. Permite conocer las redes Wi-Fi disponibles.
- CHANGE_WIFI_STATE – Cambiar estado de Wi-Fi. Permite cambiar el estado de conectividad Wi-Fi.
- CHANGE_WIFI_MULTICAST_STATE – Cambiar estado multicast Wi-Fi (API 4). Permite pasar al modo Wi-Fi Multicast.


Bluetooth:

- BLUETOOTH – Crear conexión Bluetooth. Permite a una aplicación conectarse con otro dispositivo Bluetooth. Antes ambos dispositivos han de emparejarse.



- BLUETOOTH_ADMIN - Emparejar Bluetooth. Permite descubrir y emparejarse con otros dispositivos Bluetooth.

 Consumo de batería:

- WAKE_LOCK - Impedir que el teléfono entre en modo de suspensión. Para algunas aplicaciones, como un navegador GPS, puede ser importante que no sean suspendidas nuca. Realmente, a lo único que puede afectar es a nuestra batería.
- FLASHLIGHT - Linterna. Permite encender el flash de la cámara.
- VIBRATE - Control de la vibración. Permite hacer vibrar al teléfono. Los juegos suelen utilizarlo.

 Aplicaciones:

- RECEIVE_BOOT_COMPLETED - Ejecución automática al encender el teléfono. Permite a una aplicación recibir el anuncio broadcast `ACTION_BOOT_COMPLETED` enviado cuando el sistema finaliza un inicio. Gracias a esto la aplicación pondrá ponerse en ejecución al arrancar el teléfono.
- BROADCAST_STICKY - Enviar anuncios broadcast permanentes. Un broadcast permanente llegará a los receptores de anuncios que actualmente estén escuchando, pero también a los que se instancien en un futuro. Por ejemplo, el sistema emite el anuncio broadcast `ACTION_BATTERY_CHANGED` de forma permanente. De esta forma, cuando se llama a `registerReceiver()` se obtiene la intención de la última emisión de este anuncio. Por lo tanto, puede usarse para encontrar el estado de la batería sin necesidad de esperar a un futuro cambio en su estado. Se ha incluido este permiso dado que las aplicaciones mal intencionadas pueden ralentizar el dispositivo o volverlo inestable al demandar demasiada memoria.
- KILL_BACKGROUND_PROCESSES - Matar procesos en Background (API 9). Permite llamar a `killBackgroundProcesses(String)`. Al hacer esta llamada el sistema mata de inmediato a todos los procesos de fondo asociados con el paquete indicado. Es el mismo método que usa el sistema cuando necesita memoria. Estos procesos serán reiniciados en el futuro, cuando sea necesario.
- REORDER_TASKS - Reordenar tareas. Permite a una aplicación cambiar el orden de la lista de tareas.
- INSTALL_SHORTCUT y UNINSTALL_SHORTCUT - Instalar y desinstalar acceso directo (API 19). Permite a una aplicación añadir o eliminar un acceso directo a nuestra aplicación en el escritorio.
- GET_PACKAGE_SIZE - Obtener tamaño de un paquete. Permite a una aplicación conocer el tamaño de cualquier paquete.
- EXPAND_STATUS_BAR - Expandir barra de estado. Permite a una aplicación expandir o contraer la barra de estado.
- FOREGROUND_SERVICE - Crear servicios en primer plano. (API 28). Permite a una aplicación crear servicios en primer plano.

 Configuraciones del sistema:

- SET_WALLPAPER – Poner fondo de pantalla. Permite establecer fondo de pantalla en el escritorio.
- SET_WALLPAPER_HINTS – Sugerencias de fondo de pantalla. Permite a las aplicaciones establecer sugerencias del fondo de pantalla.
- SET_ALARM – Establecer Alarma. Permite a la aplicación enviar una intención para poner una alarma o temporizador en la aplicación Reloj.
- SET_TIME_ZONE – Cambiar zona horaria. Permite cambiar la zona horaria.
- ACCESS_NOTIFICATION_POLICY – Acceso a política de notificaciones (API 23). Permite conocer la política de notificaciones del sistema.

 Audio:

- MODIFY_AUDIO_SETTINGS – Cambiar ajustes de audio. Permite cambiar ajustes globales de audio, como el volumen.

 Sincronización:

- READ_SYNC_SETTINGS – Leer ajustes de sincronización. Permite saber si tienes sincronización en segundo plano con alguna aplicación (como con un cliente de Twitter o Gmail).
- WRITE_SYNC_SETTINGS – Escribir ajustes de sincronización. Permite registrar tu aplicación como adaptador de sincronización (`SyncAdapter`).
- READ_SYNC_STATS – Leer estadísticas de sincronización.

 Ubicación:

- ACCESS_LOCATION_EXTRA_COMMANDS – Mandar comandos extras de localización. Permite a una aplicación acceder a comandos adicionales de los proveedores de localización. Por ejemplo, tras pedir este permiso podríamos enviar el siguiente comando al GPS, con el método: `sendExtraCommand("gps", "delete_aiding_data", null);`.

 Seguridad:

- USE_FINGERPRINT – Usar huella digital (API 23). Permite usar el hardware de reconocimiento de huella digital.
- DISABLE_KEYGUARD – Deshabilitar bloqueo de teclado. Permite a las aplicaciones desactivar el bloqueo del teclado si no es seguro.

NOTA: *Los permisos peligrosos pertenecen a uno de los 9 grupos anteriores. Estos grupos son importantes dado que el usuario concede o deniega el permiso a un grupo entero. Por el contrario, a partir de la versión 6.0 los permisos normales ya no se clasifican en grupos. Se han organizado en este texto por grupos para una mejor organización.*

NOTA: *Existen otros permisos que no han sido incluidos en esta lista dado que no podemos solicitarlos en nuestras aplicaciones al estar reservados para aplicaciones del sistema.*

Para solicitar un determinado permiso en tu aplicación, no tienes más que incluir una etiqueta <uses-permission> en el fichero AndroidManifest.xml de tu aplicación. En el siguiente ejemplo se solicitan dos permisos:

```
<manifest package="org.example.mi_aplicacion" >
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    …
</manifest>
```



Preguntas de repaso: *Los permisos en Android*

7.1.4. Permisos desde Android 6 Marshmallow

En un dispositivo con versión de Android anterior a Marshmallow un usuario concede los permisos a una aplicación en el momento de la instalación. Si no está de acuerdo con algún permiso, la única alternativa para el usuario es no instalar la aplicación. Una vez instalada la aplicación, puede realizar las acciones asociadas a estos permisos tantas veces como desee y cuando desee. Esta forma de trabajar dejaba a los usuarios indefensos ante posibles abusos. Por ejemplo, si queremos utilizar WhatsApp o jugar a Apalabrados tenemos que aceptar la larga lista de permisos innecesarios que nos solicitan. El usuario se resigna y acaba aceptando prácticamente cualquier permiso.

En la versión 6 se introducen importantes novedades a la hora de conceder los permisos a las aplicaciones. En primer lugar, los permisos son divididos en normales y peligrosos. A su vez los permisos peligrosos se dividen en 9 grupos: almacenamiento, localización, teléfono, SMS, contactos, calendario, cámara, micrófono y sensor de ritmo cardíaco. En el proceso de instalación el usuario da el visto bueno a los permisos normales, de la misma forma como se hacía en la versión anterior. Por el contrario, los permisos peligrosos no son concedidos en la instalación. La aplicación consultará al usuario si quiere conceder un permiso peligroso en el momento de utilizarlo:



Además, se recomienda que la aplicación indique para qué lo necesita. De esta forma el usuario tendrá más elementos de juicio para decidir si da o no el permiso. Si el usuario no concede el permiso la aplicación ha de tratar de continuar el proceso sin este permiso. Otro aspecto interesante es que el usuario podrá configurar en cualquier momento qué permisos concede y cuáles no. Por ejemplo, podemos ir al administrador de aplicaciones y seleccionar la aplicación Navegador. En el apartado Permisos se nos mostrará los grupos de permisos que podemos conceder:

Observa como de los grupos de permisos solicitados, en este momento solo concedemos el permiso de Ubicación.

El usuario concede o rechaza los permisos por grupos. Si en el manifiesto se ha pedido leer y escribir en la SD, concedemos los dos permisos o ninguno. Es decir, no podemos conceder permiso de lectura, pero denegar el de escritura.



Vídeo[tutorial]: *Permisos en Android 6.0 Marshmallow*

Para reforzar los conceptos que acabamos de exponer, es recomendable que hagas el siguiente ejercicio:



Ejercicio: *Trabajando con permisos en Android 6*

1. Crea un nuevo proyecto con los siguientes datos:

```
Phone and Tablet / Scrolling Activity
Name: Permisos en Marshmallow
Package name: org.example.permisosenmarshmallow
Language: Java ó Kotlin
Minimum API level: API 19 Android 4.4 (KitKat)
```

2. En el método onCreate() de la actividad principal, elimina las líneas tachadas, y en su lugar, añade las subrayadas.

```
borrarLlamada();
Snackbar.make(view, "Replace with your …", Snackbar.LENGTH_LONG)
        .setAction("Action", null).show();
```

3. En la etiqueta <ConstraintLayout> de *content_main.xml* añade:

```
android:id="@+id/vista_principal"
```

4. **En Java** declara la siguiente variable al principio de la clase:

```
private View vista_principal;
```

En el método onCreate() añade:

```
vista_principal = findViewById(R.id.vista_principal);
```

5. Añade el siguiente método:

```
void borrarLlamada() {
  getContentResolver().delete(CallLog.Calls.CONTENT_URI,
                                        "number='555555555'", null);
  Snackbar.make(vista_principal, "Llamadas borradas del registro.",
                                        Snackbar.LENGTH_SHORT).show();
}
```

```
fun borrarLlamada() {
    contentResolver.delete(CallLog.Calls.CONTENT_URI,
                         "number='555555555'", null)
    Snackbar.make(vista_principal, "Llamadas borradas del registro.",
          Snackbar.LENGTH_SHORT).show()
}
```

Como se describirá en el capítulo 9, este código elimina del registro de llamadas del teléfono todas las llamadas cuyo número sea 555555555. La segunda línea muestra un cuadro de texto tipo *Snackbar* para avisar que la acción se ha realizado.

6. Observa cómo el sistema nos advierte de que estamos actuando de forma no correcta:

```
getContentResolver().delete(CallLog.Calls.CONTENT_URI, "nu
Call requires permission which may be rejected by user: code should explicitly check to see if
permission is available (with 'checkPermission') or explicitly handle a potential
'SecurityException' more... (Ctrl+F1)
```

7. Ignora esta advertencia y ejecuta el proyecto. Si pulsas en el botón flotante, aparecerá el siguiente error:

Se ha detenido la aplicación Permisos en Marshmallow.

ACEPTAR

8. Abre el *LogCat* para verificar la causa del error:

```
Caused by: java.lang.SecurityException: Permission Denial: opening
   provider com.android.providers.contacts.CallLogProvider from … requires
   android.permission.READ_CALL_LOG or android.permission.WRITE_CALL_LOG
```

Es decir, la aplicación se ha detenido porque está realizando una acción que requiere de la solicitud de un permiso.

9. Añade en *AndroidManifest.xml*:

```
<uses-permission android:name="android.permission.WRITE_CALL_LOG"/>
```

10. Si ejecutas de nuevo el proyecto en un dispositivo con una versión anterior a la 6.0, podrás verificar que ya no se produce el error.

11. Si ejecutas ahora en un dispositivo con versión 6.0 o superior (si no dispones de uno utiliza un emulador), observarás que el error continúa.

12. Para entender lo que ha ocurrido, ve a *Ajustes / Aplicaciones / Permisos en Marshmallow* / Permisos:

21:19
Permisos de aplicaciones
Permisos en Marshmallow
Teléfono

Desde aquí podrás configurar los permisos peligrosos que quieres otorgar a la aplicación. Observa como el grupo de permisos referentes al teléfono está desactivado. Cuando instalamos una aplicación no se le concede ningún permiso peligroso.

13. Activa el permiso:

Teléfono

14. *Vuelve* a ejecutar la aplicación y verificar que ya no se produce el error.

Como acabamos de comprobar la aplicación anterior va a funcionar *correctamente* en dispositivos con una versión anterior a la 6.0. Sin embargo, *cuando* se ejecute en las nuevas versiones, se producirá un error. Aunque hemos *visto* cómo el usuario puede evitarlo, no es desde luego la forma correcta de trabajar.

A partir de Android Marshmallow trabajar con acciones que necesiten de un permiso va a suponer un esfuerzo adicional para el programador. Antes de realizar la acción tendremos que verificar si tenemos el permiso. En caso negativo hay que exponer al usuario para qué lo queremos y pedírselo. Si el usuario no nos diera el permiso, tendremos qué decidir qué hacer. ¿Podemos realizar la acción solicitada aunque no dispongamos de cierta información? ¿Dejamos de hacer la acción solicitada? ¿O salimos de la aplicación? En el siguiente ejercicio veremos cómo realizar esta tarea.



Vídeo[tutorial]: *Trabajando con permisos en Android 6.0*



Ejercicio: *Solicitud de permisos en Android Marshmallow*

1. El primer paso va a ser verificar que tenemos el permiso adecuado antes de realizar una acción que lo requiera. Resulta sencillo, simplemente has de añadir el `if` que se muestra a continuación en `borrarLlamada()`:

```
if (ActivityCompat.checkSelfPermission(this,Manifest.permission
              .WRITE_CALL_LOG) == PackageManager.PERMISSION_GRANTED) {
    getContentResolver().delete(CallLog.Calls.CONTENT_URI,
                                          "number='555555555'", null);
    Snackbar.make(vista_principal, "Llamadas borradas del registro.",
                                          Snackbar.LENGTH_SHORT).show();
}
```

2. Ejecuta de nuevo la aplicación en un dispositivo con versión 6.0 o superior y verifica que ya no se produce el error.

3. Esto no resuelve el problema. Nuestra aplicación no puede limitarse a no realizar la acción cuando no disponga del permiso. Ha de avisar al usuario y solicitar el permiso. Para ello añade una sección `else` a el `if` anterior:

```
if (ActivityCompat.checkSelfPermission(this, Manifest.permission
             .WRITE_CALL_LOG)== PackageManager.PERMISSION_GRANTED) {
   …
} else {
   solicitarPermiso(Manifest.permission.WRITE_CALL_LOG, "Sin el permiso"+
      " administrar llamadas no puedo borrar llamadas del registro.",
      SOLICITUD_PERMISO_WRITE_CALL_LOG, this);
}
```

4. Añade el siguiente método:

```
public static void solicitarPermiso(final String permiso, String
        justificacion, final int requestCode, final Activity actividad) {
   if (ActivityCompat.shouldShowRequestPermissionRationale(actividad,
                                                            permiso)){
      new AlertDialog.Builder(actividad)
         .setTitle("Solicitud de permiso")
         .setMessage(justificacion)
         .setPositiveButton("Ok", new DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int whichButton) {
               ActivityCompat.requestPermissions(actividad,
                     new String[]{permiso}, requestCode);
            }}).show();
   } else {
      ActivityCompat.requestPermissions(actividad,
            new String[]{permiso}, requestCode);
   }
}
```

```
fun solicitarPermiso(permiso: String, justificacion: String,
                     requestCode: Int, actividad: Activity) {
   if (ActivityCompat.shouldShowRequestPermissionRationale(actividad,
                                                            permiso)) {
      AlertDialog.Builder(actividad)
            .setTitle("Solicitud de permiso")
            .setMessage(justificacion)
            .setPositiveButton("Ok", DialogInterface.OnClickListener {
               dialog, whichButton -> ActivityCompat.requestPermissions(
                  actividad, arrayOf(permiso), requestCode )
            }).show()
   } else {
      ActivityCompat.requestPermissions(
            actividad,arrayOf(permiso), requestCode)
   }
}
```

Es posible que tengas que solicitar permisos desde diferentes puntos de la aplicación. Por esta razón se ha declarado este método público y estático. Además, se ha pasado a parámetros toda la información que necesita: el permiso a solicitar, la justificación de por qué lo necesitamos, un código de solicitud y la actividad que recogerá la respuesta. Una vez el usuario decida si da el permiso, se llamará al método `onRequestPermissionsResult()`, que tendrás que declarar en la actividad que se pasa en el cuarto parámetro. El código es un valor numérico que permitirá identificar diferentes solicitudes.

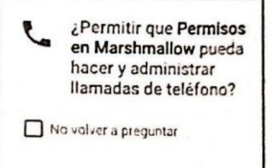
Android nos recomienda que indiquemos al usuario para qué le estamos solicitando el permiso. Si consideras que no es necesario, puedes eliminar la primera parte del método y dejar solo el código que aparece dentro del `else`. Antes de mostrar la explicación usando un `AlertDialog`, se verifica en el `if` si interesa mostrar esta información. Si el usuario ha indicado que no nos da el permiso y además ha marcado la casilla de que no quiere que volvamos a preguntar, no es conveniente insistir. El sistema se encarga de recordar esta

información, nosotros simplemente tenemos que usar el método `shouldShowRequestPermissionRationale()`.

Sin permiso para administrar llamadas no puedo borrar llamadas del registro. OK

NOTA: Este código se ejecuta en el hilo principal, por lo tanto, nunca utilices un método para preguntar al usuario que pueda bloquear el hilo. Observa como en el ejemplo se utilizan llamadas asíncronas.

El trabajo más importante lo hace el método `requestPermissions()` que muestra un cuadro de diálogo como el siguiente y registra el permiso según la respuesta del usuario:



5. Una vez que el usuario escoja se realizará una llamada a `onRequestPermissionsResult()`. Aquí podremos procesar la respuesta. Añade el siguiente método:

```
@Override public void onRequestPermissionsResult(int requestCode,
                           String[] permissions, int[] grantResults) {
   if (requestCode == SOLICITUD_PERMISO_WRITE_CALL_LOG) {
      if (grantResults.length == 1 &&
            grantResults[0] == PackageManager.PERMISSION_GRANTED) {
         borrarLlamada();
      } else {
         Toast.makeText(this, "Sin el permiso, no puedo realizar la " +
                        "acción", Toast.LENGTH_SHORT).show();
      }
   }
}
```

```
override fun onRequestPermissionsResult(requestCode: Int,
                  permissions: Array<String>, grantResults: IntArray) {
   if (requestCode == SOLICITUD_PERMISO_WRITE_CALL_LOG) {
      if (grantResults.size == 1 &&
            grantResults[0] == PackageManager.PERMISSION_GRANTED) {
         borrarLlamada()
      } else {
         Toast.makeText(this, "Sin el permiso, no puedo realizar la " +
                        "acción", Toast.LENGTH_SHORT).show()
      }
   }
}
```

Este método debe estar declarado en una actividad. En caso de que el usuario nos conceda el permiso, tenemos que volver a realizar la acción que no pudo realizarse (en el ejemplo, `borrarLlamada()`). En caso de solicitar el permiso desde diferentes acciones o solicitar diferentes permisos, el valor de `requestCode` permitirá diferenciar cada caso.

6. Declara la siguiente constante al principio de la clase:

```
private static final int SOLICITUD_PERMISO_WRITE_CALL_LOG = 0;
```

```
val SOLICITUD_PERMISO_WRITE_CALL_LOG = 0
```

7. Verifica que la aplicación funciona correctamente.



Práctica: *Solicitud de permisos en Mis Lugares*

En el ejercicio *Intenciones implícitas en Mis Lugares* se utilizó una intención asociada a `ACTION_DIAL` para realizar una llamada de teléfono. No fue necesario solicitar permiso, dado que la intención no llega a realizar la llamada. Solo marca el teléfono y es el usuario quien confirma la llamada.

Reemplaza `ACTION_DIAL` por `ACTION_CALL`. Ahora la llamada se realizará directamente sin que el usuario la confirme y, por lo tanto, esta acción sí que se considera peligrosa. Para verificar que es así, añade el permiso correspondiente en AndroidManifest y asigna el permiso manualmente en *Ajustes / Aplicaciones / Mis Lugares* / Permisos. Ejecuta la aplicación y verifica que la llamada se hace directamente.

Introduce el código para que verifique el permiso y, si es necesario, se solicite al usuario, tal y como se ha realizado en el ejercicio anterior.



Preguntas de repaso: *Permisos en Android 6.0 Marshmallow*

7.1.5. Permisos definidos por el programador en Android

Además de los permisos definidos por el sistema, los desarrolladores vamos a poder crear nuevos permisos para restringir el acceso a elementos de nuestro *software*.

***NOTA:** Se trata de un aspecto avanzado no necesario en la mayoría de aplicaciones.*



Vídeo[**tutorial**]: *Permisos definidos por el usuario en Android*

Abordaremos el estudio de la creación de nuevos permisos utilizando el siguiente ejemplo. Somos la empresa PayPerView, especializada en ofrecer

servicios de reproducción de vídeos bajo demanda. Queremos crear un *software* que permita a cualquier desarrollador reproducir nuestros vídeos desde sus aplicaciones. No obstante, este servicio no es gratuito, por lo que nos interesa que el usuario sea advertido cuando se instale la aplicación del tercero, indicándole que esta aplicación va a hacer uso de un servicio no gratuito.

Para definir el nuevo permiso utilizaremos el *tag* `<permission>` en el fichero `AndroidManifest.xml` de nuestro *software*. A continuación se muestra un ejemplo:

```
<permission
   android:name="com.payperview.servicios.VER_VIDEOS"
   android:description="@string/descripcion"
   android:label="@string/etiqueta"
   android:permissionGroup="android.permission-group.COST_MONEY"
   android:protectionLevel="dangerous" />
```

El atributo `android:name` indica el nombre del permiso. Como ves, ha de estar dentro del mismo dominio que nuestra aplicación. El atributo `android:permissionGroup` es opcional y permite incluir nuestro permiso en un grupo. En el ejemplo se ha incluido en el grupo de permisos que pueden suponer un coste económico al usuario. El atributo `android:protectionLevel` informa al sistema de cómo ha de ser informado el usuario y qué aplicaciones tienen acceso a la funcionalidad protegida. A continuación se indican los valores posibles:

`normal`	El usuario no es advertido de que se va a utilizar el permiso cuando instala la aplicación.
`dangerous`	El usuario es advertido en el proceso de instalación.
`signature`	Solo se da el permiso a aplicaciones firmadas con la misma firma digital que la aplicación que declara el permiso.
`signatureOrSystem`	Igual que `signature`, pero además puede ser usado por el sistema. Caso poco frecuente, donde varios fabricantes necesitan compartir características a través del sistema.

Los atributos `android:label` y `android:description` son opcionales y han de ser introducidos a través de un recurso de cadena. En estas cadenas hay que describir el permiso de forma abreviada y extensa, respectivamente. Veamos cómo podría ser en el ejemplo:

```
<string name="etiqueta">
   reproducción de vídeos bajo demanda </string>
<string name="descripcion">Permite a la aplicación reproducir
   vídeos de la empresa PayPerView sin tu intervención. Se trata
   de un servicio no gratuito, por lo que puede afectar al saldo
   de tu cuenta con esta empresa. Si no tienes una cuenta
   abierta los vídeos no podrán ser reproducidos. </string>
```



Ejercicio: *Creando tus propios permisos*

1. Crea un nuevo proyecto con los siguientes datos:

```
Phone and Tablet / Empty Activity
Name: Nuevo Permiso
Package name: com.payperview.servicios
Language: Java
Minimum API level: API 19 Android 4.4 (KitKat)
```

2. Crea una nueva actividad en este proyecto que se llame `VerVideo` y copia el mismo código de la actividad principal. En el ejemplo real, esta nueva actividad sería la responsable de la reproducción de vídeos.
3. En lugar de visualizar un vídeo vamos a poner un *toast*. Añade en el método `onCreate` el siguiente código:

```
Toast.makeText(this, "Reproduciendo Vídeo", Toast.LENGTH_SHORT).show();
```

La aplicación NuevoPermiso va a tener dos actividades: `MainActivity` y `VerVideo`. Para abreviar el ejemplo, estas actividades no hacen nada, se limitan a poner *"Hello Word"*. La actividad `VerVideo` es la que reproduciría los vídeos y la que queremos proteger con un permiso. La actividad `MainActivity` no sirve para nada, existe para que la aplicación tenga una actividad principal.

4. Modifica el fichero `AndroidManifest.xml` según el código mostrado a continuación:

```
<manifest…>
  <application…>
    …
    <activity
      android:name="VerVideo"
      android:permission= "com.payperview.servicios.VER_VIDEOS">
      <intent-filter>
        <action android:name="android.intent.action.VIEW" />
      </intent-filter>
    </activity>
  </application>
</manifest>
```

5. Copia antes de `<application…/>` la etiqueta `<permission …/>` del ejemplo anterior.
6. Recuerda definir los recursos de cadena `etiqueta` y `descripcion`, tal y como se indica en el ejemplo anterior.
7. Ejecuta el proyecto. Es imprescindible para registrar en el teléfono el nuevo permiso y la nueva actividad que queremos lanzar desde otras aplicaciones.
8. Para usar este servicio crea un nuevo proyecto con los siguientes datos:

```
Phone and Tablet / Empty Activity
Name: Usar Permiso
Package name: org.example.usarpermiso
Language: Java
Minimum API level: API 19 Android 4.4 (KitKat)
```

9. Abre el fichero del *layout* principal (`activity_main.xml`) y reemplaza el código por el siguiente:

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  xmlns:tools="http://schemas.android.com/tools"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical" >
  <Button
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:onClick="verVideo"
     android:text="Ver Vídeo" />
</LinearLayout>
```

10. Abre la actividad principal y añade el siguiente método:

```
public void verVideo (View view){
   Intent i = new Intent();
   i.setClassName("com.payperview.servicios",
                  "com.payperview.servicios.VerVideo");
   startActivity(i);
}
```

11. Ejecuta la aplicación. Cuando pulses el botón la aplicación provocará un error.
12. Visualiza la ventana *LogCat* para verificar que se trata de un error de permiso. Ha de aparecer algo parecido a:

```
java.lang.SecurityException: Permission Denial: starting Inten
t { cmp=com.payperview.servicios/.VerVideo } from ProcessRecor
d{439b3bd0 270:com.example.usarpermiso/10035} (pid=270, uid=10
035) requires com.payperview.servicios.VER_VIDEOS
```

13. Para solucionar el problema tendrás que incluir el siguiente código al principio del fichero `AndroidManifest.xml`:

```
<uses-permission android:name="com.payperview.servicios.VER_VIDEOS"/>
```

14. Comprueba que ahora se accede al servicio sin problemas.
15. En este ejercicio hemos puesto el nivel de protección del permiso como `dangerous`. No obstante, si la aplicación se instala desde Android Studio no se nos advierte sobre este permiso. Para que sí lo advierta, tenemos que instalar la aplicación manualmente.

Para ello, con un administrador de archivo, abre la carpeta *bin* del proyecto y copia el fichero *UsarPermiso.apk* en la memoria externa del dispositivo. Instala la aplicación. Verás como aparece un mensaje similar al siguiente:



7.1.6. Cambios relacionados con la privacidad en Android 9

Con el objeto de mejorar la privacidad de usuario, Android 9 introduce varios cambios de comportamiento, entre los que se incluyen nuevas reglas de permisos y grupos de permisos relacionados con las llamadas telefónicas, el estado del teléfono y análisis de WiFi. Estos cambios afectan a todas las aplicaciones que se ejecuten en Android 9, sin importar la versión de SDK a la que se orienten.

Entre las nuevas medidas, Android 9 restringe el acceso a los registros de llamadas. Para ello, se introduce el grupo de permisos `CALL_LOG`, y se mueven al mismo los permisos `READ_CALL_LOG`, `WRITE_CALL_LOG` y `PROCESS_OUTGOING_CALLS`. En versiones anteriores de Android, estos permisos se encontraban en el grupo de permisos `PHONE`. Este nuevo grupo de permisos brinda a los usuarios más control y visibilidad respecto de las aplicaciones que necesiten acceder a información confidencial sobre llamadas telefónicas, como puede ser la lectura de registro de llamadas y la identificación de los números telefónicos.

Al mismo tiempo, se ha modificado la política de Google Play y solo las aplicaciones por defecto tendrán acceso al registro de SMS y de llamadas, esto es, aplicaciones cuya única finalidad sea el envío y recepción de SMS o llamadas. De acuerdo a esto, aplicaciones como WhatsApp, que piden acceso a los SMS y al teléfono del usuario de cara a su activación, tendrían que eliminar dichos permisos ya que no los utilizan como base de su funcionamiento. Eso sí, al tratarse de una política de la Google Play, aquellas aplicaciones que se descarguen fuera de Google Play no tendrán que asumir la restricción. Incluso aunque se descarguen desde dentro del mismo, al no tratarse de una medida implementada en el sistema, los desarrolladores podrían continuar con las peticiones, bajo el riesgo de ser expulsados de Google Play.

7.1.7. Cambios relacionados con la privacidad en Android 10

En la versión 10 también se introducen importantes novedades relacionadas con la privacidad. La más importante es el ámbito de almacenamiento (Scoped Storage), que consiste en que las aplicaciones solo pueden ver el contenido de las carpetas creadas por ellas. Gracias a esto ya no va a ser necesario pedir permiso para acceder al almacenamiento externo, si accedemos a ficheros creados por la aplicación.

Otra novedad es que se añaden capas encima del sistema de permisos de Android 9. Se pretende que permisos sensibles como micrófono y localización solo sean válidos cuando la app está en primer plano. Por ejemplo, si queremos que una

aplicación pueda localizarnos en segundo plano, hemos de darle el nuevo permiso `ACCESS_BACKGROUND_LOCATION`.

Se añaden nuevas restricciones. Las apps deben tener permiso de firma `READ_PRIVILEGE_STATE_STORAGE` para acceder a los identificadores del dispositivo que no se pueden restablecer, incluido IMEI y número de serie.

Tampoco podrán habilitar/deshabilitar WiFi. `WifiManager.setWifiEnabled()` siempre mostrará `false`. Para esto será obligatorio usar el panel de configuración.

Se debe tener el permiso `ACCESS_FINE_LOCATION` para poder usar varios métodos dentro de las API de WiFi y Bluetooth, relacionadas con localización.

Se introduce el permiso en tiempo de ejecución `ACTIVITY_RECOGNITION` para apps que necesitan detectar el recuento de pasos del usuario o clasificar su actividad física (caminar, ir en bicicleta o en un vehículo). Así los usuarios tienen mayor visibilidad sobre cómo se usan los datos del sensor del dispositivo.

7.2. Localización

La plataforma Android dispone de un interesante sistema de posicionamiento que combina varias tecnologías:

- Sistema de localización global basado en GPS. Este sistema solo funciona si disponemos de visibilidad directa de los satélites.
- Sistema de localización basado en la información recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi. Funciona en el interior de los edificios.

Estos servicios se encuentran totalmente integrados en el sistema y son usados por una gran variedad de aplicaciones. Por ejemplo, la aplicación Locale[32] de Android puede adaptar la configuración del teléfono según donde se encuentre. Podría, por ejemplo, poner el modo de llamada en vibración cuando estemos en el trabajo.

El sistema de posicionamiento global, GPS, fue diseñado inicialmente con fines militares, pero hoy en día es ampliamente utilizado para uso civil. Gracias al desfase temporal de las señales recibidas por varios de los 31 satélites desplegados, este sistema es capaz de posicionarnos en cualquier parte del planeta con una precisión de 15 metros.

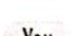
El GPS presenta un inconveniente: solo funciona cuando tenemos visión directa de los satélites. Para solventar este problema, Android combina esta información con la recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi.

[32] http://www.androidlocale.com

7.2.1. Sistemas de geolocalización en dispositivos móviles




Video[tutorial]: *Sistema de geolocalización en móviles*



Preguntas de repaso: *Sistema de geolocalización en móviles*




Video[tutorial]: *Los sistemas de posicionamiento global por satélite*



Preguntas de repaso: *GPS*



7.2.2. La API de localización de Android



Video[tutorial]: *La API de localización de Android*

Ejercicio: *La API de localización de Android*

En este ejercicio crearemos una aplicación que es capaz de leer información de localización del dispositivo y actualizarla cada vez que se produce un cambio.

1. Crea un nuevo proyecto con los siguientes datos:

```
Phone and Tablet / Empty Activity
Name: Localizacion
Package name: org.example.localizacion
Language: Java o Kotlin
Minimum API level: API 19 Android 4.4 (KitKat)
```

2. Por razones de privacidad, acceder a la información de localización en principio está prohibido a las aplicaciones. Si estas desean hacer uso de este servicio han de solicitar el permiso adecuado. En concreto, hay que solicitar `ACCESS_FINE_LOCATION` para acceder a cualquier tipo de sistema de localización o `ACCESS_COARSE_LOCATION` para acceder al sistema de localización basado en

redes. Añade la siguiente línea en el fichero *AndroidManifest.xml* dentro de la etiqueta `<manifest>`:

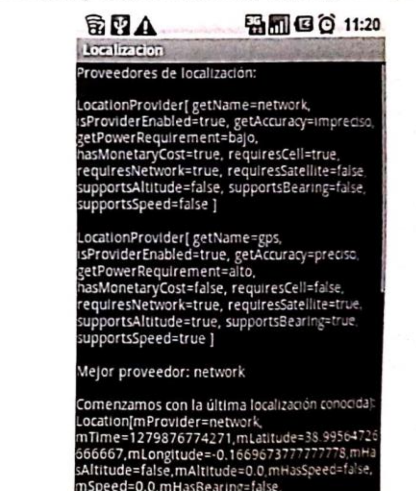
```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Por lo tanto, en este ejemplo vamos a utilizar tanto la localización fina que nos proporciona el GPS como una menos precisa que nos proporcionan las torres de telefonía celular y las redes Wi-Fi.

3. Sustituye el fichero `res/layout/activity_main.xml` por:

```
<ScrollView
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent">
   <TextView
      android:id="@+id/salida"
      android:layout_width="match_parent"
      android:layout_height="wrap_content" />
</ScrollView>
```

En este ejemplo nos limitaremos a mostrar en modo de texto la información obtenida desde la API de localización. Para ello usaremos un `TextView` dentro de un `ScrollView`, tal y como se muestra en la siguiente pantalla:



4. Abre la clase `MainActivity` y copia el siguiente código:

```
public class MainActivity extends AppCompatActivity
                 implements LocationListener {
  static final long TIEMPO_MIN = 10 * 1000 ; // 10 segundos
  static final long DISTANCIA_MIN = 5; // 5 metros
  static final String[] A = { "n/d", "preciso", "impreciso" };
```

```
  static final String[] P = { "n/d", "bajo", "medio","alto" };
  static final String[] E = { "fuera de servicio",
                  "temporalmente no disponible ", "disponible" };
  LocationManager manejadorLoc;
  String proveedor;
  TextView salida;

  @Override public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.activity_main);
     salida = findViewById(R.id.salida);
     manejadorLoc = (LocationManager) getSystemService(LOCATION_SERVICE);
     log("Proveedores de localización: \n ");
     muestraProveedores();

     Criteria criterio = new Criteria();
     criterio.setCostAllowed(false);
     criterio.setAltitudeRequired(false);
     criterio.setAccuracy(Criteria.ACCURACY_FINE);
     proveedor = manejadorLoc.getBestProvider(criterio, true);
     log("Mejor proveedor: " + proveedor + "\n");
     log("Comenzamos con la última localización conocida:");
     Location localizacion= manejadorLoc.getLastKnownLocation(proveedor);
     muestraLocaliz(localizacion);
  }
```

```
class MainActivity : AppCompatActivity(), LocationListener {

  val TIEMPO_MIN = (10 * 1000).toLong() // 10 segundos
  val DISTANCIA_MIN = 5.0F // 5 metros
  val A = arrayOf("n/d", "preciso", "impreciso")
  val P = arrayOf("n/d", "bajo", "medio", "alto")
  val E = arrayOf("fuera de servicio", "temporalmente no disponible",
          "disponible")
  lateinit var manejadorLoc: LocationManager
  lateinit var proveedor: String

  public override fun onCreate(savedInstanceState: Bundle?) {
     super.onCreate(savedInstanceState)
     setContentView(R.layout.activity_main)
     manejador = getSystemService(Context.LOCATION_SERVICE) as
                                                  LocationManager
     log("Proveedores de localización: \n ")
     muestraProveedores()
     val criterio = Criteria().apply {
        isCostAllowed = false
        isAltitudeRequired = false
        accuracy = Criteria.ACCURACY_FINE
     }
     proveedor = manejadorLoc.getBestProvider(criterio, true)
     log("Mejor proveedor: $proveedor\n")
     log("Comenzamos con la última localización conocida:")
     muestraLocaliz(manejadorLoc.getLastKnownLocation(proveedor))
  }
```


La primera línea que nos interesa es la llamada a `getSystemService(LOCATION_SERVICE)`, que crea el objeto `manejadorLoc` de tipo `LocationManager`. La siguiente línea hace una llamada al método `log()`, que se definirá más adelante. Simplemente muestra por el `TextView`, `salida`, el texto indicado. La siguiente llamada a `muestraProveedores()` también es un método definido por nosotros, que listará todos los proveedores de localización disponibles.

En las siguientes líneas vamos a seleccionar uno de estos proveedores de localización. Comenzamos creando un objeto de la clase `Criteria`, donde se podrán indicar las características que ha de tener el proveedor buscado. En este ejemplo indicamos que no ha de tener coste económico, ha de poder obtener la altura y ha de tener precisión fina. Para consultar otras restricciones, véase documentación de la clase `Criteria`[33]. Con estas restricciones parece que estamos interesados en el proveedor basado en GPS, aunque de no estar disponible, se seleccionará otro que cumpla el mayor número de restricciones. Para seleccionar el proveedor usaremos el método `getBestProvider()`. En este método hay que indicar el criterio de selección y un valor booleano, donde indicamos si solo nos interesan los sistemas que el usuario tenga actualmente habilitados. Nos devolverá un `String` con el nombre del proveedor seleccionado.

Algunos proveedores, como el GPS, pueden tardar cierto tiempo en darnos una primera posición. No obstante, Android recuerda la última posición que fue devuelta por ese proveedor. Es lo que nos devuelve la llamada a `getLastKnownLocation()`. El método `muestraLocaliz()` se definirá más tarde y muestra en pantalla una determinada localización.

5. A continuación, copia el resto del código:

```
// Métodos del ciclo de vida de la actividad
@Override protected void onResume() {
   super.onResume();
   manejadorLoc.requestLocationUpdates(proveedor, TIEMPO_MIN,
                                              DISTANCIA_MIN, this);
}

@Override protected void onPause() {
   super.onPause();
   manejadorLoc.removeUpdates(this);
}

// Métodos de la interfaz LocationListener
@Override public void onLocationChanged(Location location) {
   log("Nueva localización: ");
   muestraLocaliz(location);
}
```

[33] http://developer.android.com/reference/android/location/Criteria.html

```java
@Override public void onProviderDisabled(String proveedor) {
   log("Proveedor deshabilitado: " + proveedor + "\n");
}

@Override public void onProviderEnabled(String proveedor) {
   log("Proveedor habilitado: " + proveedor + "\n");
}

@Override public void onStatusChanged(String proveedor, int estado,
                                              Bundle extras) {
   log("Cambia estado proveedor: " + proveedor + ", estado="
         + E[Math.max(0, estado)] + ", extras=" + extras + "\n");
}

// Métodos para mostrar información
private void log(String cadena) {
   salida.append(cadena + "\n");
}

private void muestraLocaliz(Location localizacion) {
   if (localizacion == null)
      log("Localización desconocida\n");
   else
      log(localizacion.toString() + "\n");
}

private void muestraProveedores() {
   log("Proveedores de localización: \n ");
   List<String> proveedores = manejadorLoc.getAllProviders();
   for (String proveedor : proveedores) {
      muestraProveedor(proveedor);
   }
}

private void muestraProveedor(String proveedor) {
   LocationProvider info = manejadorLoc.getProvider(proveedor);
   log("LocationProvider[ " + "getName=" + info.getName()
         + ", isProviderEnabled="
         + manejadorLoc.isProviderEnabled(proveedor) + ", getAccuracy="
         + A[Math.max(0, info.getAccuracy())] + ", getPowerRequirement="
         + P[Math.max(0, info.getPowerRequirement())]
         + ", hasMonetaryCost=" + info.hasMonetaryCost()
         + ", requiresCell=" + info.requiresCell()
         + ", requiresNetwork=" + info.requiresNetwork()
         + ", requiresSatellite=" + info.requiresSatellite()
         + ", supportsAltitude=" + info.supportsAltitude()
         + ", supportsBearing=" + info.supportsBearing()
         + ", supportsSpeed=" + info.supportsSpeed() + " ]\n");
}
}
```

```kotlin
   override fun onPause() {
      super.onPause()
      manejadorLoc.removeUpdates(this)
   }
```



```kotlin
// Métodos de la interfaz LocationListener
override fun onLocationChanged(location: Location) {
   log("Nueva localización: ")
   muestraLocaliz(location)
}

override fun onProviderDisabled(proveedor: String) {
   log("Proveedor deshabilitado: $proveedor\n")
}

override fun onProviderEnabled(proveedor: String) {
   log("Proveedor habilitado: $proveedor\n")
}

override fun onStatusChanged(proveedor: String, estado: Int,
                                              extras: Bundle) {
   log("Cambia estado proveedor: $proveedor, estado="+
              " ${E[Math.max(0, estado)]}, extras= $extras\n")
}

// Métodos para mostrar información
private fun log(cadena: String) = salida.append(cadena + "\n")

private fun muestraLocaliz(localizacion: Location?) {
   if (localizacion == null)
      log("Localización desconocida\n")
   else
      log(localizacion!!.toString() + "\n")
}

private fun muestraProveedores() {
   log("Proveedores de localización: \n ")
   val proveedores = manejadorLoc.getAllProviders()
   for (proveedor in proveedores) {
      muestraProveedor(proveedor)
   }
}

private fun muestraProveedor(proveedor: String) {
   with(   manejadorLoc.getProvider(proveedor)) {
      log("LocationProvider[ " + "getName= $name, isProviderEnabled" +
         "=${manejadorLoc.isProviderEnabled(proveedor)}, " +
         "getAccuracy=${A[Math.max(0, accuracy)]}, " +
         "getPowerRequirement=${P[Math.max(0, powerRequirement)]}, " +
         "hasMonetaryCost=${hasMonetaryCost()}, " +
         "requiresCell=${requiresCell()}, " +
         "requiresNetwork=${requiresNetwork()}, " +
         "requiresSatellite=${requiresSatellite()}, " +
         "supportsAltitude=${supportsAltitude()}, " +
         "supportsBearing=${supportsBearing()}, " +
         "supportsSpeed=${supportsSpeed()} ]\n")
   }
 }
}
```

Para conseguir que se notifiquen cambios de posición hay que llamar al método `requestLocationUpdates()` y para indicar que se dejen de hacer las notificaciones hay que llamar a `removeUpdates()`. Dado que queremos ahorrar batería, nos interesa que se reporten notificaciones solo cuando la aplicación esté activa. Por lo tanto, tenemos que rescribir los métodos `onResume()` y `onPause()`.

El método `requestLocationUpdates()` dispone de 4 parámetros: el nombre del proveedor, el tiempo entre actualizaciones en ms (se recomiendan valores superiores a 60.000 ms), la distancia mínima (de manera que, si es menor, no se notifica) y un escuchador de eventos que implemente la interfaz `LocationListener`.

Como nuestra actividad es un `LocationListener`, tenemos que implementar el método `onLocationChanged()`, que se activará cada vez que se obtenga una nueva posición. Los otros tres métodos pueden ser usados para cambiar de proveedor en caso de que se active uno mejor o deje de funcionar el actual. Sería buena idea llamar de nuevo aquí al método `getBestProvider()`.

El resto del código resulta fácil de interpretar.

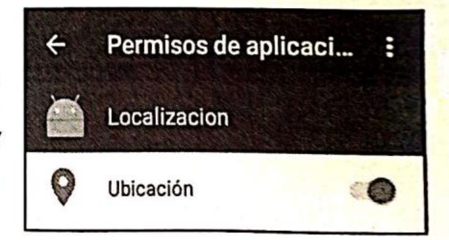
6. Observa cómo aparecen algunos errores. El sistema nos advierte de que estamos actuando de forma no correcta:

```
Location localizacion = manejador.getLastKnownLocation(proveedor);
```

Call requires permission which may be rejected by user: code should explicitly check to see if permission is available (with `checkPermission`) or explicitly handle a potential `SecurityException` more... (Ctrl+F1)

7. Ignora esta advertencia y ejecuta el proyecto.

8. Si ejecutas el proyecto en un dispositivo con una versión 6.0 o superior, podrás verificar que se produce el error. Para evitar que se produzca, en el dispositivo accede a *Ajustes / Aplicaciones / Localización / Permisos* y activa el permiso de *Ubicación*:



Vuelve a ejecutar la aplicación y verificar que ya no se produce el error.

NOTA: *Para aligerar el código del ejercicio no hemos incluido el código necesario para solicitar permiso a partir de la versión 6.0. Si vas a distribuir la aplicación, resulta imprescindible realizar los pasos descritos en la sección Permisos en Android 6 Marshmallow.*

9. Verifica el funcionamiento del programa, si es posible con un dispositivo real con el GPS activado.



Preguntas de repaso: *API Localización de Android*

Usar siempre el mismo tipo de proveedor

Los dos proveedores de localización disponibles en Android tienen características muy diferentes. Muchas aplicaciones tienen algún tipo de requisito que hace que podamos decantarnos de entrada por un sistema en concreto. Veamos algunos ejemplos.

Usaremos GPS si:

- La aplicación requiere una precisión inferior a 10 m.
- Está pensada para su uso al aire libre (p. ej., senderismo).

Usaremos localización por redes si:

- El consumo de batería es un problema.
- Está pensada para su uso en el interior de edificios (visita museo).

Una vez decidido, usaremos las constantes `GPS_PROVIDER` o `NETWORK_PROVIDER` de la clase `LocationManager` para indicar el proveedor deseado.

Existe un tercer tipo de proveedor identificado con la constante `PASSIVE_PROVIDER`. Puedes usarlo si quieres observar pasivamente actualizaciones de ubicación provocadas por otras aplicaciones, pero no quieres que se lancen nuevas lecturas de posición. De esta manera no provocamos consumo de energía adicional.

El mejor proveedor según un determinado criterio

Como vimos en el apartado anterior, la API de localización de Android nos proporciona la clase `Criteria` para seleccionar un proveedor de localización según el criterio indicado. Recordemos el código utilizado:

```
Criteria criterio = new Criteria();
criterio.setCostAllowed(false);
criterio.setAltitudeRequired(false);
criterio.setAccuracy(Criteria.ACCURACY_FINE);
proveedor = manejadorLoc.getBestProvider(criterio, true);
```

Los proveedores pueden variar de estado, por lo que podría ser interesante consultar cuál es el mejor proveedor cada vez que cambie su estado.

Usar los dos proveedores en paralelo

Otra alternativa podría ser programar actualizaciones de los dos proveedores de localización disponibles. Luego podríamos seleccionar la mejor localización entre las suministradas. Para estudiar esta alternativa realiza el siguiente ejercicio:



Vídeo[tutorial]: *Estrategias de localización en Android*



Ejercicio: *Añadiendo localización en Mis Lugares*

1. Añade en *AndroidManifest.xml* de Mis Lugares el siguiente permiso:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

2. Crea la clase `CasosUsoLocalizacion` con los siguientes atributos y su inicialización:

```
public class CasosUsoLocalizacion {
   private static final String TAG = "MisLugares";
   private Activity actividad;
   private int codigoPermiso;
   private LocationManager manejadorLoc;
   private Location mejorLoc;
   private GeoPunto posicionActual;
   private AdaptadorLugares adaptador;

   public CasosUsoLocalizacion(Activity actividad, int codigoPermiso) {
      this.actividad = actividad;
      this.codigoPermiso = codigoPermiso;
      manejadorLoc = (LocationManager) actividad.getSystemService(
                                                      LOCATION_SERVICE);
      posicionActual = ((Aplicacion) actividad.getApplication())
                                                      .posicionActual;
      adaptador = ((Aplicacion) actividad.getApplication()).adaptador;
      ultimaLocalizazion();
   }
}
```

```
class CasosUsoLocalizacion(val actividad: Activity,
                           val codigoPermiso: Int) : LocationListener {
   val TAG = "MisLugares"
   val manejadorLoc = actividad.getSystemService(
                  AppCompatActivity.LOCATION_SERVICE) as LocationManager
   var mejorLoc: Location? = null
   val posicionActual = (actividad.application as Aplicacion).posicionActual
   val adaptador = (actividad.application as Aplicacion).adaptador

   init {
      ultimaLocalizazion()
   }
}
```

La variable `manejadorLoc` nos permite acceder a los servicios de localización de Android. La variable `mejorLoc`, de tipo `Location`, almacena la mejor localización actual. La variable `posicionActual` almacena la misma información, pero en formato `GeoPunto`. Estará almacenada en `Aplicacion` para que sea accesible desde cualquier parte de la aplicación. Es necesario tener la información en dos formatos, ya que la primera variable es usada para disponer de la fecha de obtención o proveedor que nos la ha dado y la segunda al ser el formato usado en el resto de la aplicación. Finalmente obtenemos una referencia al adaptador del `RecyclerView` para poder actualizarlo cuando haya cambios de localización.

3. En la clase `Aplicacion` crea la variable `posicionActual`:

```
public GeoPunto posicionActual =  new GeoPunto(0.0, 0.0);
```

```
val posicionActual = GeoPunto.SIN_POSICION
```

Los valores (0, 0) representa que no se dispone de localización.

4. En `MainActivity`, añade:

```
private static final int SOLICITUD_PERMISO_LOCALIZACION = 1;
private CasosUsoLocalizacion usoLocalizacion;


@Override protected void onCreate(Bundle savedInstanceState) {
   …
   usoLocalizacion = new CasosUsoLocalizacion(this,
                                      SOLICITUD_PERMISO_LOCALIZACION);
}
```

```
val SOLICITUD_PERMISO_LOCALIZACION = 1
val usoLocalizacion by lazy {
   CasosUsoLocalizacion(this, SOLICITUD_PERMISO_LOCALIZACION) }
```

5. Vamos a verificar varias veces si el usuario nos ha dado permiso de localización. Para ello añade en `CasosUsoLocalizacion` el siguiente código:

```
public boolean hayPermisoLocalizacion() {
   return (ActivityCompat.checkSelfPermission(
           actividad, Manifest.permission.ACCESS_FINE_LOCATION)
           == PackageManager.PERMISSION_GRANTED);
}
```

```
fun hayPermisoLocalizacion() = (ActivityCompat.checkSelfPermission(
   actividad, Manifest.permission.ACCESS_FINE_LOCATION)
         == PackageManager.PERMISSION_GRANTED)
```

6. Añade el siguiente método:

```
void ultimaLocalizazion() {
  if (hayPermisoLocalizacion()) {
    if (manejadorLoc.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
         actualizaMejorLocaliz(manejadorLoc.getLastKnownLocation(
             LocationManager.GPS_PROVIDER));
    }
    if (manejadorLoc.isProviderEnabled(LocationManager.NETWORK_PROVIDER)){
         actualizaMejorLocaliz(manejadorLoc.getLastKnownLocation(
             LocationManager.NETWORK_PROVIDER));
    }
  } else {
         solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,
             "Sin el permiso localización no puedo mostrar la distancia"+
             " a los lugares.", codigoPermiso, actividad);
  }
}
```



```
@SuppressLint("MissingPermission")
fun ultimaLocalizazion() {
  if (hayPermisoLocalizacion()) {
    if (manejadorLoc.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
         actualizaMejorLocaliz( manejadorLoc.getLastKnownLocation(
                LocationManager.GPS_PROVIDER))
    }
    if (manejadorLoc.isProviderEnabled(LocationManager.NETWORK_PROVIDER)){
         actualizaMejorLocaliz( manejadorLoc.getLastKnownLocation(
                LocationManager.NETWORK_PROVIDER))
    }
  } else {
         solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,
            "Sin el permiso localización no puedo mostrar la distancia" +
            " a los lugares.", codigoPermiso, actividad)
  }
}
```

Antes de obtener una localización se debe verificar que tenemos permiso para hacerlo. Para más información consultar *Permisos en Android 6 Marshmallow*. En caso de tener permiso buscamos la última localización disponible. Usamos el método `getLastKnownLocation()` aplicado a los dos proveedores que vamos a utilizar. El método `actualizaMejorLocaliz()` se explicará más adelante. Si no disponemos del permiso, lo solicitamos al usuario.

7. La función `getLastKnownLocation()` estará marcada con el error *"Call requiered permision …"*, avisándonos que hemos de comprobar que tenemos permiso antes de hacer la llamada. Realmente lo hemos hecho. Para desactivar la advertencia añade `@SuppressLint("MissingPermission")` antes de la función.

8. Copia a esta clase el método `solicitarPermiso()` del ejercicio *Permisos en Android 6 Marshmallow*.

9. Una vez conteste el usuario se llamará a `onRequestPermissionsResult` de `MainActivity`. Añade en `MainActivity` el siguiente método:

```
@Override public void onRequestPermissionsResult(int requestCode,
                                String[] permissions, int[] grantResults) {
   if (requestCode == SOLICITUD_PERMISO_LOCALIZACION
      && grantResults.length == 1
      && grantResults[0] == PackageManager.PERMISSION_GRANTED)
     usoLocalizacion.permisoConcedido();
}
```

```
override fun onRequestPermissionsResult(requestCode: Int,
              permissions: Array<String>, grantResults: IntArray ) {
   if (requestCode == SOLICITUD_PERMISO_LOCALIZACION
      && grantResults.size == 1
      && grantResults[0] == PackageManager.PERMISSION_GRANTED)
    usoLocalizacion.permisoConcedido()
}
```

Si el usuario contesta afirmativamente a la solicitud de permiso llamamos a un caso de uso para que se actúe en consecuencia.

10. Añade el siguiente caso de uso y función asociada en `CasosUsoLocalizacion`:

```
public void permisoConcedido() {
   ultimaLocalizazion();
   activarProveedores();
   adaptador.notifyDataSetChanged();
}

@SuppressLint("MissingPermission")
private void activarProveedores() {
  if (hayPermisoLocalizacion()) {
    if (manejadorLoc.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
         manejadorLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
              20 * 1000, 5, this);
    }
    if (manejadorLoc.isProviderEnabled(LocationManager.NETWORK_PROVIDER)){
              manejadorLoc.requestLocationUpdates(LocationManager
              .NETWORK_PROVIDER, 10 * 1000, 10, this);
      }
  } else {
     solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,
              "Sin el permiso localización no puedo mostrar la distancia"+
              " a los lugares.", codigoPermiso, actividad);
  }
}
```

```
fun permisoConcedido() {
   ultimaLocalizazion()
   activarProveedores()
   adaptador.notifyDataSetChanged()
}

@SuppressLint("MissingPermission")
private fun activarProveedores() {
  if (hayPermisoLocalizacion()) {
    if (manejadorLoc.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
         manejadorLoc.requestLocationUpdates(
            LocationManager.GPS_PROVIDER,20 * 1000, 5F, this )
    }
    if (manejadorLoc.isProviderEnabled(LocationManager.NETWORK_PROVIDER)){
         manejadorLoc.requestLocationUpdates(
            LocationManager.NETWORK_PROVIDER, 10 * 1000, 10F, this )
      }
  } else {
     solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,
         "Sin el permiso localización no puedo mostrar la distancia" +
         " a los lugares.", codigoPermiso, actividad )
  }
}
```

La primera función se llama cuando nos conceden permiso de localización. Miramos si ya disponen de una última posición conocida, llamamos a la segunda función que activa los eventos de localización y refrescamos el `RecyclerView`.

La segunda función hace que nuestra clase (`this`) sea informada con cada actualización del proveedor de localización. Lo hacemos para el proveedor basado en GPS (cada 10s y si hay un cambio de más de 5m) y con el basado en redes (cada 20s y si hay un cambio de más de 10m).

11. Para recibir los eventos de localización haz que la clase `CasosUsoLocalizacion` implemente la interfaz `LocationListener` y añade las siguientes funciones:

```
@Override public void onLocationChanged(Location location) {
   Log.d(TAG, "Nueva localización: "+location);
   actualizaMejorLocaliz(location);
   adaptador.notifyDataSetChanged();
}
@Override public void onProviderDisabled(String proveedor) {
   Log.d(TAG, "Se deshabilita: "+proveedor);
   activarProveedores();
}
@Override public void onProviderEnabled(String proveedor) {
   Log.d(TAG, "Se habilita: "+proveedor);
   activarProveedores();
}
@Override
public void onStatusChanged(String proveedor, int estado, Bundle extras) {
   Log.d(TAG, "Cambia estado: "+proveedor);
   activarProveedores();
}
```

```
override fun onLocationChanged(location: Location) {
   Log.d(TAG, "Nueva localización: $location")
   actualizaMejorLocaliz(location)
   adaptador.notifyDataSetChanged()
}
override fun onProviderDisabled(proveedor: String) {
   Log.d(TAG, "Se deshabilita: $proveedor")
   activarProveedores()
}
override fun onProviderEnabled(proveedor: String) {
   Log.d(TAG, "Se habilita: $proveedor")
   activarProveedores()
}
override fun onStatusChanged(proveedor:String, estado:Int, extras:Bundle){
   Log.d(TAG, "Cambia estado: $proveedor")
   activarProveedores()
}
```

Las acciones a realizar resultan evidentes: cuando la actualizamos cambie la posición y cuando cambie el estado tratamos de activar nuevos proveedores.

12. Ahora añade la siguiente función:

```
private static final long DOS_MINUTOS = 2 * 60 * 1000;

private void actualizaMejorLocaliz(Location localiz) {
   if (localiz != null && (mejorLoc == null
```

```
        || localiz.getAccuracy() < 2*mejorLoc.getAccuracy()
        || localiz.getTime() - mejorLoc.getTime() > DOS_MINUTOS)) {
      Log.d(TAG, "Nueva mejor localización");
      mejorLoc = localiz;
      posicionActual.setLatitud(localiz.getLatitude());
      posicionActual.setLongitud(localiz.getLongitude());
   }
}
```

```
val DOS_MINUTOS:Long = (2 * 60 * 1000)

private fun actualizaMejorLocaliz(loc: Location?) {
   if (localiz != null && (mejorLoc == null
            || loc.accuracy < 2 * mejorLoc!!.getAccuracy()
            || loc.time - mejorLoc!!.getTime() > DOS_MINUTOS)) {
      Log.d(TAG, "Nueva mejor localización")
      mejorLoc = loc
      posicionActual.latitud = loc.latitude
      posicionActual.longitud = loc.longitude
   }
}
```

En la variable `mejorLoc` almacenamos la mejor localización. Esta solo será actualizada con la nueva propuesta si: todavía no ha sido inicializada; o la nueva localización tiene una precisión aceptable (al menos la mitad que la actual); o la diferencia de tiempo es superior a dos minutos. Una vez comprobado si se cumple alguna de las tres condiciones, actualizamos `mejorLocaliz` y copiamos la posición en `posicionActual`.

13. Si dejáramos activos los escuchadores de eventos mientras la aplicación está en segundo plano, podríamos quedarnos sin batería. Para evitar esta situación añade en `MainActivity`:

```
@Override protected void onResume() {
   super.onResume();
   usoLocalizacion.activar();
}

@Override protected void onPause() {
   super.onPause();
   usoLocalizacion.desactivar();
}
```

```
override fun onResume() {
   super.onResume()
   usoLocalizacion.activar()
}

override fun onPause() {
   super.onPause()
   usoLocalizacion.desactivar()
}
```


14. Añade los dos nuevos casos de uso:

```
public void activar() {
   if (hayPermisoLocalizacion()) activarProveedores();
}

public void desactivar() {
   if (hayPermisoLocalizacion()) manejadorLoc.removeUpdates(this);
}
```

```
fun activar() {
   if (hayPermisoLocalizacion()) activarProveedores()
}

fun desactivar() {
   if (hayPermisoLocalizacion()) manejadorLoc.removeUpdates(this)
}
```

15. Una vez que ya disponemos de la posición actual, vamos a tratar de mostrar la distancia a cada lugar en el `RecyclerView` de la actividad principal. Abre el *layout elemento_lista.xml* y añade al final del `ConstraintLayout` la nueva vista que se indica:

```
…
<TextView
   android:id="@+id/distancia"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_toRightOf="@id/valoracion"
   android:gravity="right"
   android:text="... Km"
   app:layout_constraintEnd_toEndOf="parent"
   app:layout_constraintTop_toBottomOf="@+id/direccion" />

</…ConstraintLayout>
```

16. Para Java en `AdaptadorLugares`, dentro del `ViewHolder`, añade la variable:

```
public TextView distancia;
```

En el constructor de `ViewHolder` añade al final:

```
distancia = itemView.findViewById(R.id.distancia);
```

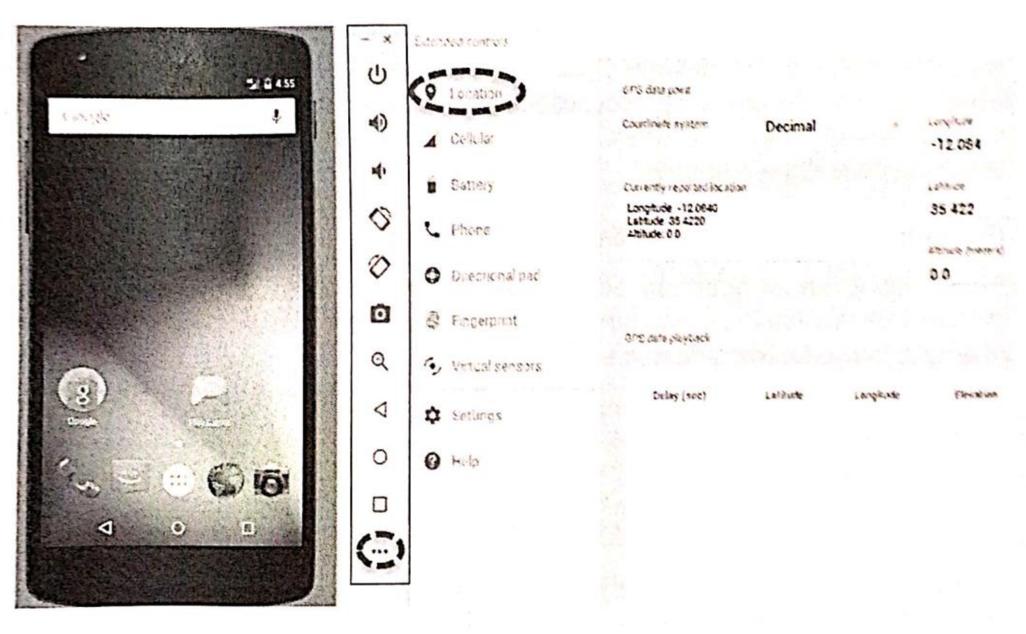
17. Dentro del método `personaliza()` añade el siguiente código al final:

```
GeoPunto pos=((Aplicacion) itemView.getContext().getApplicationContext())
                                                        .posicionActual;
if (pos.equals(GeoPunto.SIN_POSICION) ||
                 lugar.getPosicion().equals(GeoPunto.SIN_POSICION)) {
   distancia.setText("... Km");
} else {
   int d=(int) pos.distancia(lugar.getPosicion());
   if (d < 2000) distancia.setText(d + " m");
   else          distancia.setText(d / 1000 + " Km");
}
```

7.2.3. Emulación del GPS con Android Studio

Probar una aplicación de localización con un dispositivo real requiere que te desplaces para cambiar tu posición. Suele resultar más práctico probar este tipo de aplicaciones desde un emulador, ya que estos incorporan un sistema de emulación de posición GPS. Para abrir los controles extendidos de un emulador pulsa en los tres puntos que aparecen en la parte inferior de la barra de herramientas y selecciona la pestaña *Location*.

Desde aquí podremos enviar nuevas posiciones al dispositivo que está siendo emulado. El botón *LOAD GPX / KML* nos permiten realizar pruebas más complejas en nuestras aplicaciones de localización, sin necesidad de dar vueltas con el dispositivo en la mano. Un fichero GPX o KML registra una secuencia temporal de localizaciones. Existen muchos programas (Google Earth) que permiten grabar este tipo de ficheros. Luego podremos reproducir esta secuencia tantas veces como queramos hasta que nuestro programa funcione perfectamente.





Práctica: *Emulación del GPS*

Ejecuta el proyecto anterior en un emulador y prueba la emulación del GPS.

7.2.4. Estrategias para escoger un proveedor de localización

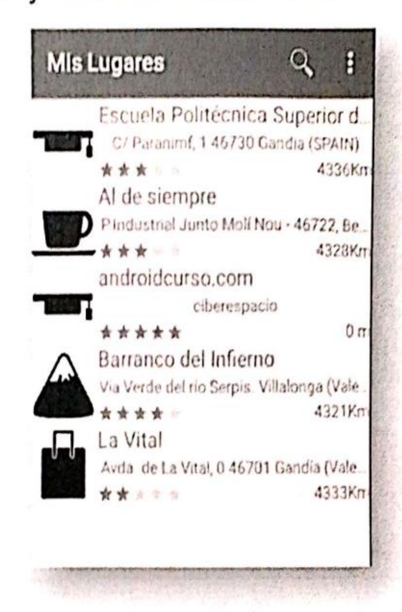
Determinar cuál es el proveedor de localización idóneo para nuestra aplicación puede resultar una tarea compleja. Además, esta decisión puede variar con el tiempo según el usuario cambie de posición, o puede desactivarse alguno de los proveedores. A continuación, se plantean tres posibles estrategias.

```
val pos = (context.applicationContext as Aplicacion).posicionActual
if (pos==GeoPunto.SIN_POSICION || lugar.posicion==GeoPunto.SIN_POSICION) {
   distancia.text = "... Km"
} else {
   val d = pos.distancia(lugar.posicion).toInt()
   distancia.text = if (d < 2000) "$d m"
                    else          "${(d / 1000)} Km"

}
```

Nos aseguramos de que la posición actual y la del lugar existen. Luego calculamos la distancia en la variable `d`. Si la distancia es inferior a 2000, se muestra en metros; en caso contrario se muestra en Km.

18. Ejecuta la aplicación y verifica el resultado obtenido:



NOTA: En este ejercicio se ha decidido extraer todo el código que nos permite mantener la localización del dispositivo a una nueva clase. Se podría haber integrado dentro de `MainActivity`, como se hizo en el ejercicio "La API de localización de Android". Hacerlo de esta forma divide las responsabilidades entre las dos clases, lo que las hace más fáciles de entender y de mantener. Además, el código es más reutilizable, si en un futuro queremos que otra actividad acceda a la localización, podremos usar la clase `CasosUsoLocalizacion` sin tener que cambiarla.

7.2.5. Límites de ubicación en segundo plano

En un esfuerzo por reducir el consumo de energía, Android 8.0 limitó la frecuencia con la cual las aplicaciones en segundo plano pueden recuperar la ubicación actual del usuario, sin importar la versión de SDK de destino de tu aplicación. Este comportamiento de obtención de la ubicación es particularmente importante para tener en cuenta si tu aplicación depende de alertas o detección de movimiento en tiempo real mientras se ejecuta en segundo plano.

El sistema distingue entre las aplicaciones en primer y segundo plano. Se considera que una aplicación se encuentra en primer plano si alguno de los siguientes puntos se cumple:

- Tiene una actividad visible, independientemente de que la actividad se haya iniciado o esté en pausa.
- Tiene un servicio en primer plano (muestra una notificación para la barra de estado).
- Otra aplicación en primer plano está conectada a la aplicación, ya sea por vinculación a uno de sus servicios o por el uso de uno de sus proveedores de contenido.

Si ninguna de estas condiciones se cumple, se considera que la aplicación se encuentra en segundo plano, y las actualizaciones de la ubicación se ven reducidas a sólo algunas veces por hora. En caso contrario, si la aplicación se encuentra en primer plano, el comportamiento de la actualización de la ubicación es el mismo que en versiones de Android anteriores a la 8.0.

7.3. Google Maps

Google Maps nos proporciona un servicio de cartografía *online* que podremos utilizar en nuestras aplicaciones Android. Entre las ventajas que aporta destaca el menor tráfico intercambiado con el servidor, la utilización de *fragments* y los gráficos en 3D. Como inconveniente cabe resaltar que la nueva versión solo funciona en el dispositivo con Google Play instalado.

Conviene destacar que, a diferencia de Android, Google Maps no es un *software* libre, por lo que está limitado a una serie de condiciones de servicio. Desde Julio de 2018 se ha introducido una serie de restricciones de uso[34] que debemos tener en cuenta:

- Mientras la política de utilización anterior nos permitía centralizar 18 APIs de localización distintas, ahora se han reducido a 3: mapas, rutas y lugares.
- Google “regalará” 200 dólares mensuales de uso a cada desarrollador que utilice las nuevas APIs de Google Maps.
- El acceso a Google Maps se integra dentro de la plataforma Cloud de Google. Esto obliga a los desarrolladores a indicar un medio de pago para utilizar las APIs, aunque no vaya a exceder de los 200 dólares mensuales de crédito.
- Las llamadas gratuitas a las APIs se han limitado de 25.000 peticiones diarias a 28.000 por mes.
- Para poder utilizar el API de Google Maps el desarrollador deberá tener una llave válida, actualizada y su perfil de Google Cloud deberá incluir, como indicamos previamente, sus datos bancarios.

A cambio de lo anterior, podemos incluir propaganda en los mapas o incluso podemos usarlo en aplicaciones móviles de pago.

---

[34] https://cloud.google.com/maps-platform/pricing/sheet/?hl=es

7.3.1. Obtención de una clave Google Maps

Para poder utilizar este servicio de Google, igual que como ocurre cuando se utiliza desde una página web, será necesario registrar la aplicación que lo utilizará. Tras registrar la aplicación se nos entregará una clave que tendremos que indicar en la aplicación.



Ejercicio: *Obtención de una clave Google Maps*

1. Para obtener la clave Google Maps entra en la página web de Google Cloud: https://cloud.google.com/console/
2. Tendrás que introducir un usuario de Google que realiza la solicitud.
3. Crea un nuevo proyecto en esta consola. Para ello, abre el desplegable de la parte superior y en la ventana emergente selecciona *NUEVO PROYECTO*. Introduce como nombre *Ejemplo Google Maps* y pulsa *Crear* (el proceso tardará algunos segundos y debes tener en cuenta que no podrás modificar el nombre asignado al proyecto).
4. Una vez generado, selecciónalo en el desplegable superior y accede a la opción *Ir a la visión general de las APIs,* que encontrarás dentro de la tarjeta APIs.
5. En la nueva pantalla podrás ver algunos accesos directos a las APIs más comunes, entre los que se suele encontrar la opción de *Maps SDK for Android*. Si no es así, selecciona la opción *HABILITAR APIS Y SERVICIOS* en la parte superior, donde podrás acceder a *Maps SDK for Android*.
6. Una vez dentro de la opción de mapas podrás encontrar información relacionada con la documentación o el listado de precios[35]. Te recomendamos que leas detenidamente ambas informaciones. Una vez hecho, pulsa en *Habilitar*.
7. En la nueva ventana podremos consultar información relevante sobre las cuotas de uso y métricas de nuestra aplicación, una vez esté operativa. De momento, selecciona la pestaña de *Credenciales*. En la ventana emergente selecciona *Crear Credenciales*. Selecciona *Clave de API*.
8. En la ventana siguiente, copia al portapapeles la clave creada:

Clave de API creada

Para usar esta clave en tu aplicación, transfiérela como un parámetro
key=API_KEY

Tu clave de API
AIzaSyCfcwo2LEBJ11aiW3bxZ9tUujYULXI-GN8

Restringe la clave para impedir el uso no autorizado en producción.

CERRAR RESTRINGIR CLAVE

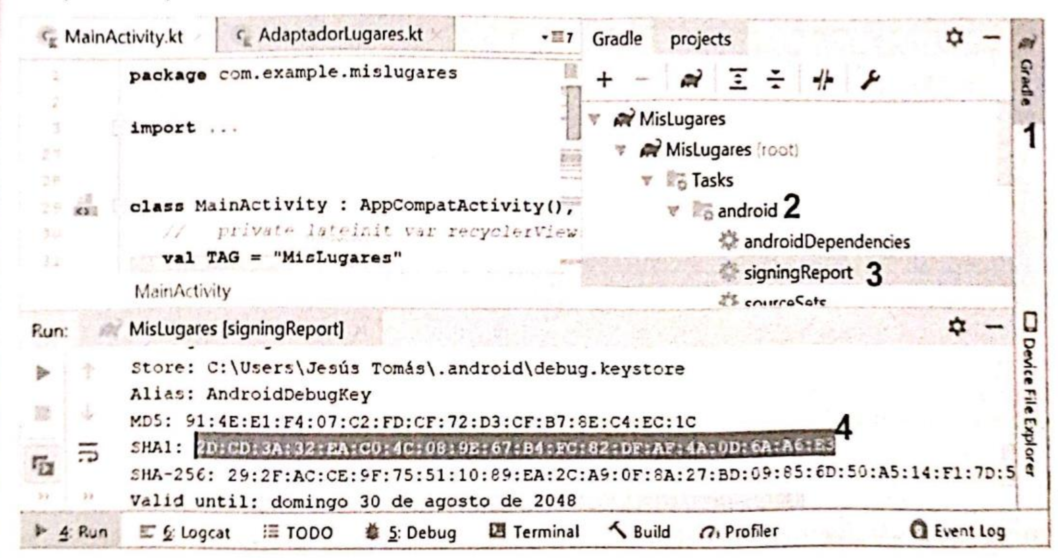
[35] https://developers.google.com/maps/billing/understanding-cost-of-use?hl=es#maps-product



Ejercicio: *Restringir uso de la clave Google Maps*

La clave que acabamos de crear podría caer en malas manos y ser usada desde otra aplicación Android, iOS o Web. Esto podría ser perjudicial para nosotros, al verse reducida la cuenta asignada a esta clave. Una forma de evitar usos no autorizados de esta clave consiste en indicar a Google que solo permita el uso de esta clave desde nuestra aplicación. Para conseguirlo, lo primero que necesitamos es la huella digital SHA1 del certificado digital con el que se ha firmado nuestra aplicación. En esta fase de desarrollo estamos usando el certificado digital de depuración. En la fase de publicación, el certificado será diferente y tendremos que volver a obtener la huella SHA1. Los pasos 1 al 4 permiten obtener la huella digital SHA1 desde Android Studio. Los pasos 5 al 9 realizan lo mismo, pero desde la línea de comando. Puedes utilizar la alternativa que prefieras.

1. Desde Android Studio, pulsa en el botón *Gradle* del panel de la derecha.
2. En el desplegable abre la ruta *<Nombre del proyecto>/Task/android.*
3. Haz doble clic en *signingReport.*
4. En la ventana *Run*, aparecerá la firma digital SHA1. Cópiala al portapapeles y pasa al punto 10.



5. El primer paso va a consistir en descubrir dónde está almacenado el certificado digital de depuración. Accede a la carpeta `.android` que encontrarás en la carpeta de tu usuario. Dentro se almacena el fichero `debug.keystore` con el certificado digital de depuración. En Windows, la ruta de este fichero podría ser `C:\Users\<Usuario>\.android\debug.keystore`. En Linux y Mac, la ruta desde la carpeta de tu usuario local es `.android/debug.keystore`.
6. Copia esta ruta en el portapapeles.

7. Ahora necesitamos extraer la huella digital SHA1 de este fichero. Para extraer la huella digital puedes utilizar el programa `keytool`. En Windows, este programa se encuentra en la carpeta `C:\Program Files\Java\jre7\bin\` o en una similar. Abre un intérprete de comandos (símbolo del sistema) y sitúate en la carpeta anterior (o similar).

```
cd C:\Archivos de programa\Java\jre7\bin
```

8. Ejecuta el siguiente comando reemplazando el nombre del fichero por el que acabas de copiar en el portapapeles.

```
keytool -v -list -keystore [ruta a debug.keystore]
```

En nuestro ejemplo:

```
keytool -v -list -keystore C:\android-sdk\.android\debug.keystore
```

```
C:\Archivos de programa\Java\jre7\bin>keytool -v -list -keystore C:\android-sdk\
.android\debug.keystore
Introduzca la contrase±a del almacÚn de claves:

*****************  WARNING WARNING WARNING  *****************
* La integridad de la informaci¾n almacenada en el almacÚn de claves  *
* NO se ha comprobado.  Para comprobar dicha integridad, *
* debe proporcionar la contrase±a del almacÚn de claves.              *
*****************  WARNING WARNING WARNING  *****************

Tipo de AlmacÚn de Claves: JKS
Proveedor de AlmacÚn de Claves: SUN

Su almacÚn de claves contiene 1 entrada

Nombre de Alias: androiddebugkey
Fecha de Creaci¾n: 20-jul-2012
Tipo de Entrada: PrivateKeyEntry
Longitud de la Cadena de Certificado: 1
Certificado[1]:
Propietario: CN=Android Debug, O=Android, C=US
Emisor: CN=Android Debug, O=Android, C=US
N·mero de serie: 7cd560d6
VBlido desde: Fri Jul 20 21:32:46 CEST 2012 hasta: Sun Jul 13 21:32:46 CEST 2042

Huellas digitales del Certificado:
         MD5: AF:7C:FC:0F:6F:68:C8:F6:40:7E:74:E3:6C:0E:69:FD
         SHA1: 9E:80:89:80:E0:54:45:AA:61:FD:38:75:E3:F5:64:08:DB:9F:83:B9
         SHA256: 0A:DC:F3:67:D9:91:93:BB:1A:8A:5E:96:12:D0:15:22:5E:72:76:57:B2:
CD:74:BC:7C:97:D4:DE:F3:7E:74:54
         Nombre del Algoritmo de Firma: SHA256withRSA
         Versi¾n: 3
```

***NOTA:** Si la ruta del fichero tiene espacios, introdúcela entre comillas.*

El programa te solicitará una contraseña para proteger el almacén de claves. Deja la contraseña en blanco. De toda la información mostrada, nos interesa la huella digital del certificado en codificación SHA1. Como puedes ver en la captura anterior, para nuestro ejemplo está formada por los siguientes bytes:

```
9E:80:89:80:E0:54:45:AA:61:FD:38:75:E3:F5:64:08:DB:9F:83:B9
```

9. Copia en el portapapeles esta secuencia de dígitos. En Windows pulsa con el botón derecho sobre la barra superior de la venta y selecciona *Marcar*. Selecciona el área a copiar. Luego, en este mismo menú, selecciona *Editar/Copiar*.



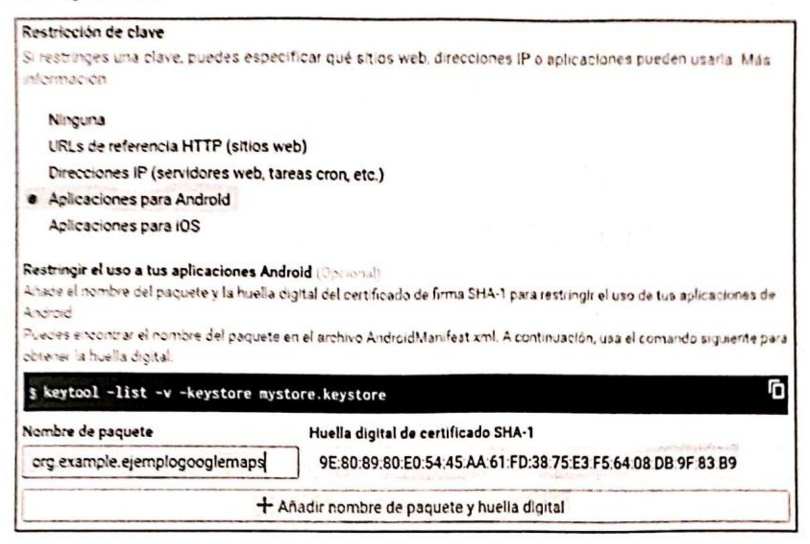
10. Accede a la consola de Google Cloud (https://cloud.google.com/console/) y selecciona el proyecto creado.

11. En el menú de la izquierda selecciona *APIs y servicios / redenciales*. En la lista de *Claves de API*, pulsa en el botón de editar (con forma de lápiz):

Claves de API

Nombre	Fecha de creación	Restricción	Clave
Clave de API 1	17 abr. 2017	Ninguna	AIzaSyCfcwo2LEBJ11aiW3bxZ9tUujYULXI-GN8

12. En Restricciones de clave, selecciona *Aplicaciones de Android* y pulsa en *Añadir nombre de paquete y huella digital*. Aparecerán dos cuadros de entrada donde has de añadir el nombre de paquete de la aplicación y la huella digital obtenida al principio del ejercicio.



13. Finalmente, pulsa en *Guardar*.



Ejercicio: *Un ejemplo simple con Google Maps*

Veamos un sencillo ejemplo que nos permite visualizar un mapa centrado en las coordenadas geográficas detectadas por el sistema de posicionamiento.

1. Crea un nuevo proyecto con los siguientes datos:

```
Phone and Tablet / Empty Activity
Name: Ejemplo Google Maps
Package name: org.example.ejemplogooglemaps
Language: Java
Minimum API level: API 19 Android 4.4 (KitKat)
```

2. Vamos a importar a nuestro proyecto el paquete de mapas de la librería Google Play Services. Para ello, añade en el graddle del módulo app la siguiente línea.

```
implementation 'com.google.android.gms:play-services-maps:17.0.0'
```

3. En *AndroidManifest.xml*, añade en siguiente permiso:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

> ***NOTA:** Los permisos de localización no son necesarios para trabajar con Google Maps, pero sí los debemos especificar para trabajar con la funcionalidad MyLocation, la cual vamos a utilizar en nuestro ejemplo. **NOTA:** Este permiso incluye de forma implícita los permisos ACCESS_COARSE_LOCATION y NETWORK_PROVIDER.*

4. Añade también las siguientes líneas dentro de la sección `<application>`:

```
<meta-data android:name="com.google.android.geo.API_KEY"
           android:value="@string/google_maps_key" />
```

5. Crea el siguiente fichero de recurso *res/values/google_maps_api.xml*:

```
<resources>
   <string name="google_maps_key" templateMergeStrategy="preserve"
                                                translatable="false">
      AIzaSyCfcwo2LEBJ11aiW3bxZ9tUujYULXI-GN8
   </string>
</resources>
```

Reemplaza los caracteres marcados (“AIza...”) por la *API Key* obtenida en el ejercicio anterior.

6. Reemplaza el contenido del *layout activity_main.xml* por:

```
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".MainActivity">
   <fragment
      android:id="@+id/mapa"
      class="com.google.android.gms.maps.SupportMapFragment"
      android:layout_width="0dp"
      android:layout_height="0dp"
      app:layout_constraintBottom_toBottomOf="parent"
      app:layout_constraintEnd_toEndOf="parent"
      app:layout_constraintStart_toStartOf="parent"
      app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

7. Abre *MainActivity* y haz que esta clase herede de `FragmentActivity` y que implemente la interfaz `OnMapReadyCallback`:

```
public class MapaActivity extends FragmentActivity
                          implements OnMapReadyCallback
```

```
class MapaActivity: FragmentActivity(), OnMapReadyCallback {
```

8. Reemplaza el método `onCreate` por el siguiente:

```
@Override protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   // Obtenemos el mapa de forma asíncrona (notificará cuando esté listo)
   SupportMapFragment mapFragment = (SupportMapFragment)
                getSupportFragmentManager().findFragmentById(R.id.mapa);
   mapFragment.getMapAsync(this);
}
```

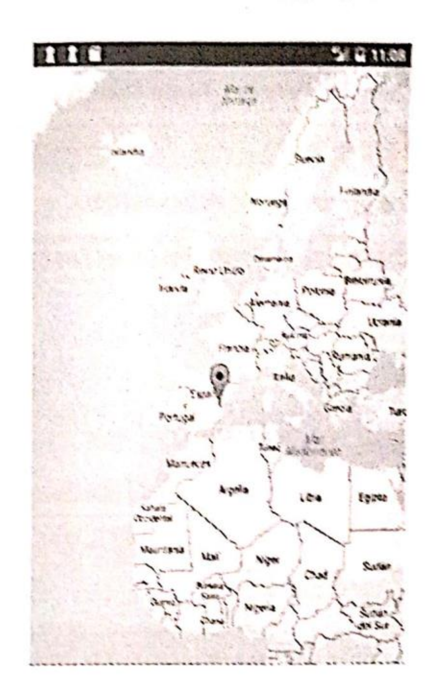
```
override fun onCreate(savedInstanceState: Bundle?) {
   super.onCreate(savedInstanceState)
   setContentView(R.layout.activity_main)
   // Obtenemos el mapa de forma asíncrona (notificará cuando esté listo)
   val mapFragment = supportFragmentManager.findFragmentById(R.id.mapa) as
                                                        SupportMapFragment
   mapFragment.getMapAsync(this)
}
```

9. Debemos implementar el método `onMapReady` que será llamado en el momento en que el mapa está disponible. Es en esta función donde podremos manipular el mapa. Una implementación mínima sería la siguiente:

```
@Override public void onMapReady(GoogleMap googleMap) {
   GoogleMap mapa = googleMap;
   LatLng UPV = new LatLng(39.481106, -0.340987); //Nos ubicamos en la UPV
   mapa.addMarker(new MarkerOptions().position(UPV).title("Marker UPV"));
   mapa.moveCamera(CameraUpdateFactory.newLatLng(UPV));
}
```

```
override fun onMapReady(googleMap: GoogleMap) {
   val UPV = LatLng(39.481106, -0.340987) //Nos ubicamos en la UPV
   googleMap.addMarker(MarkerOptions().position(UPV).title("Marker UPV"))
   googleMap.moveCamera(CameraUpdateFactory.newLatLng(UPV))
}
```

10. Ejecuta la aplicación. A continuación, se muestra el resultado:



NOTA: Si utilizas un emulador, asegúrate que disponga de servicios de Google Pla



Ejercicio: *Introduciendo código en Google Maps*

En el ejercicio anterior hemos visto un ejemplo muy básico, donde solo se mostraba un mapa con las opciones predeterminadas. En este ejercicio aprenderemos a configurarlo y añadir marcadores desde el código.

1. Abre el *layout activity_main.xml* y añade los siguientes tres botones dentro del `<ConstraintLayout>` (tras el `<fragment ...>`):

```
<Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="moveCamera"
        android:text="ir a UPV"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button2"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toStartOf="parent" />
<Button android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="animateCamera"
        android:text="animar a UPV"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/button3"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/button1" />
```



```
<Button android:id="@+id/button3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="addMarker"
        android:text="marcador"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.5"
        app:layout_constraintStart_toEndOf="@+id/button2" />
```

2. Sustituye el contenido de *MainActivity.java* por:

```
public class MainActivity extends FragmentActivity implements
                        OnMapReadyCallback, GoogleMap.OnMapClickListener {
   private GoogleMap mapa;
   private final LatLng UPV = new LatLng(39.481106, -0.340987);

   @Override protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      SupportMapFragment mapFragment = (SupportMapFragment)
               getSupportFragmentManager().findFragmentById(R.id.mapa);
      mapFragment.getMapAsync(this);
   }

   @Override public void onMapReady(GoogleMap googleMap) {
      mapa = googleMap;
      mapa.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
      mapa.getUiSettings().setZoomControlsEnabled(false);
      mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(UPV, 15));
      mapa.addMarker(new MarkerOptions()
               .position(UPV)
               .title("UPV")
               .snippet("Universidad Politécnica de Valencia")
               .icon(BitmapDescriptorFactory
                      .fromResource(android.R.drawable.ic_menu_compass))
               .anchor(0.5f, 0.5f));
      mapa.setOnMapClickListener(this);
      if (ActivityCompat.checkSelfPermission(this,
               android.Manifest.permission.ACCESS_FINE_LOCATION) ==
               PackageManager.PERMISSION_GRANTED) {
         mapa.setMyLocationEnabled(true);
         mapa.getUiSettings().setCompassEnabled(true);
      }
   }

   public void moveCamera(View view) {
      mapa.moveCamera(CameraUpdateFactory.newLatLng(UPV));
   }
   public void animateCamera(View view) {
      mapa.animateCamera(CameraUpdateFactory.newLatLng(UPV));
   }
```

```
    public void addMarker(View view) {
        mapa.addMarker(new MarkerOptions().position(
                                      mapa.getCameraPosition().target));
    }

    @Override public void onMapClick(LatLng puntoPulsado) {
        mapa.addMarker(new MarkerOptions().position(puntoPulsado)
                 .icon(BitmapDescriptorFactory
                 .defaultMarker(BitmapDescriptorFactory.HUE_YELLOW)));
    }
}
```

```
class MainActivity : FragmentActivity(), OnMapReadyCallback,
                                          GoogleMap.OnMapClickListener {
    lateinit var mapa: GoogleMap
    val UPV = LatLng(39.481106, -0.340987)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val mapFragment = supportFragmentManager.findFragmentById(R.id.mapa)
                                                  as SupportMapFragment
        mapFragment.getMapAsync(this)
    }

    override fun onMapReady(googleMap: GoogleMap) {
        mapa = googleMap.apply {
            mapType = GoogleMap.MAP_TYPE_SATELLITE
            uiSettings.isZoomControlsEnabled = false
            moveCamera(CameraUpdateFactory.newLatLngZoom(UPV, 15f))
            addMarker(MarkerOptions().position(UPV).title("UPV")
                  .snippet("Universidad Politécnica de Valencia")
                  .icon(BitmapDescriptorFactory.fromResource(
                        android.R.drawable.ic_menu_compass))
                  .anchor(0.5f, 0.5f))
        }
        if (ActivityCompat.checkSelfPermission(this,
              android.Manifest.permission.ACCESS_FINE_LOCATION)
                    == PackageManager.PERMISSION_GRANTED) {
            mapa.isMyLocationEnabled = true
            mapa.uiSettings.isCompassEnabled = true
        }
        mapa.setOnMapClickListener(this)
    }

    fun moveCamera(view: View) {
        mapa.moveCamera(CameraUpdateFactory.newLatLng(UPV))
    }

    fun animateCamera(view: View) {
        mapa.animateCamera(CameraUpdateFactory.newLatLng(UPV))
    }
```

```
    fun addMarker(view: View) {
        mapa.addMarker(MarkerOptions().position(mapa.cameraPosition.target))
    }

    override fun onMapClick(puntoPulsado: LatLng) {
        mapa.addMarker(MarkerOptions().position(puntoPulsado)
            .icon(BitmapDescriptorFactory.defaultMarker(
                BitmapDescriptorFactory.HUE_YELLOW)))
    }
}
```

Comenzamos declarando dos objetos: UPV, que hace referencia a la posición geográfica de la Universidad Politécnica de Valencia, y mapa, que nos permitirá acceder al objeto GoogleMap que hemos insertado en un *fragment* de nuestro *layout*. El mapa es cargado asíncronamiente, por lo que es necesario implementar el método onMapReady de la interfaz OnMapReadyCallback, que sera llamado en el momento en que mapa esté listo. Será en este método cuándo podremos configurar el objeto GoogleMap que nos pasan como parámetro, para adaptarlo a nuestras necesidades. setMapType() permite seleccionar el tipo de mapa (normal, satélite, hibrido o relieve). Para averiguar las constantes correspondientes, te recomendamos que utilices la opción de autocompletar (escribe GoogleMap. y podrás seleccionar las constantes de esta clase). El método moveCamera() desplaza el área de visualización a una determinada posición (UPV), a la vez que define el nivel de zum (15). El nivel de zum ha de estar en un rango de 2 (continente) hasta 21 (calle). El método addMarker() permite añadir los típicos marcadores que habrás visto en muchos mapas. En este ejemplo se indica la posición (UPV), un título, una descripción, un icono y el punto del icono, que haremos coincidir con el punto exacto que queremos indicar en el mapa. Un valor de (0, 0) corresponde a la esquina superior izquierda del icono y (1, 1), a la esquina inferior derecha. Como nuestro icono tiene forma de círculo ⊗, hemos indicado el valor (0.5, 0.5) para que coincida con su centro. Finalmente, hemos registrado un escuchador de evento para detectar pulsaciones sobre la pantalla. El escuchador vamos a ser nosotros mismos (this), por lo que hemos implementado la interfaz OnMapClickListener y añadido el método onMapClick().

El método setMyLocationEnabled(true) activa la visualización de la posición del dispositivo por medio del típico círculo. Para dispositivos con versión 6 o superior hay que tener la precaución de verificar si tenemos permiso de localización antes de activar esta opción. Por defecto, al tratarse de un permiso catalogado como peligroso se encontrará desactivado. En este código no se solicita este permiso. Para activarlo manualmente debes usar en el dispositivo la opción *Ajustes / Aplicaciones / Ejemplo Google Maps / Permisos*. El método getUiSettings() permite configurar las acciones de la interfaz de usuario. En este ejemplo se han utilizado dos: desactivar los botones de zum y visualizar una brújula. Estos métodos solo están disponibles si la capa LocationLayer está activa, por lo que es recomendable iniciarlos posteriormente al método setMyLocationEnabled(true). Puedes usar autocompletar para descubrir otras posibles configuraciones. En caso de no tener permiso de localización, debemos deshabilitar el botón que nos lleva a nuestra posición.

A continuación, se incluyen los tres métodos que se ejecutarán al pulsar sobre los botones añadidos al *layout*. El primero, `moveCamera()`, desplaza el punto de visualización a la UPV. A diferencia del uso anterior, sin cambiar el nivel de zum que el usuario tenga seleccionado.

El segundo, `animateCamera()`, nos desplaza también a la UPV por medio de una animación (similar a la que a veces utilizan en el *Telediario* para mostrar un punto en conflicto).

El tercero, `addMarker()`, añade un nuevo marcador en el centro del mapa que estamos observando (`getCameraPosition()`). En este caso usaremos el marcador por defecto, sin información adicional.

Como hemos indicado, se llamará a `onMapClick()`cuando se pulse sobre el mapa. Se pasa como parámetro las coordenadas del punto donde se ha pulsado, que utilizaremos para añadir un marcador. Esta vez el marcador será de color amarillo.

3. Ejecuta la aplicación. Se muestra el resultado.



Ejercicio: *Añadiendo Google Maps en Mis Lugares*

1. Realiza el ejercicio *"Obtención de una clave Google Maps"*, pero esta vez indica como nombre del proyecto Mis Lugares.
2. Añade la librería *Google Maps* y configura *AndroidManifest.xml*. Para ello sigue los puntos 2 al 5 del ejercicio "*Un ejemplo simple con Google Maps*".
3. Crea un nuevo *layout* que se llame *mapa.xml* con el siguiente código:

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment android:id="@+id/mapa"
              android:layout_width="match_parent"
              android:layout_height="match_parent"
              class="com.google.android.gms.maps.SupportMapFragment"/>
</LinearLayout>
```

4. Crea una nueva clase para la actividad que mostrará el mapa:

```
public class MapaActivity extends FragmentActivity
                          implements OnMapReadyCallback {
```

```
    private GoogleMap mapa;
    private RepositorioLugares lugares;

    @Override public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.mapa);
       lugares = ((Aplicacion) getApplication()).lugares;
       SupportMapFragment mapFragment = (SupportMapFragment)
                getSupportFragmentManager().findFragmentById(R.id.mapa);
       mapFragment.getMapAsync(this);
    }

    @Override public void onMapReady(GoogleMap googleMap) {
       mapa = googleMap;
       mapa.setMapType(GoogleMap.MAP_TYPE_NORMAL);
       if (ActivityCompat.checkSelfPermission(this,
              android.Manifest.permission.ACCESS_FINE_LOCATION) ==
              PackageManager.PERMISSION_GRANTED) {
          mapa.setMyLocationEnabled(true);
          mapa.getUiSettings().setZoomControlsEnabled(true);
          mapa.getUiSettings().setCompassEnabled(true);
       }
       if (lugares.tamaño() > 0) {
          GeoPunto p = lugares.elemento(0).getPosicion();
          mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(
                   new LatLng(p.getLatitud(), p.getLongitud()), 12));
       }
       for (int n=0; n<lugares.tamaño(); n++) {
          Lugar lugar = lugares.elemento(n);
          GeoPunto p = lugar.getPosicion();
          if (p != null && p.getLatitud() != 0) {
             Bitmap iGrande = BitmapFactory.decodeResource(
                           getResources(), lugar.getTipo().getRecurso());
             Bitmap icono = Bitmap.createScaledBitmap(iGrande,
                   iGrande.getWidth() / 7, iGrande.getHeight() / 7, false);
             mapa.addMarker(new MarkerOptions()
                   .position(new LatLng(p.getLatitud(), p.getLongitud()))
                   .title(lugar.getNombre()).snippet(lugar.getDireccion())
                   .icon(BitmapDescriptorFactory.fromBitmap(icono)));
          }
       }
    }
}
```

```
class MapaActivity: FragmentActivity(), OnMapReadyCallback {

   lateinit var mapa: GoogleMap
   val lugares by lazy { (application as Aplicacion).lugares }

   public override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.mapa)
      val mapFragment = supportFragmentManager.findFragmentById(R.id.mapa)
                                                    as SupportMapFragment
```

```
        mapFragment.getMapAsync(this)
    }

    override fun onMapReady(googleMap: GoogleMap) {
        mapa = googleMap
        mapa.setMapType(GoogleMap.MAP_TYPE_NORMAL)
        if (ActivityCompat.checkSelfPermission(this,
            android.Manifest.permission.ACCESS_FINE_LOCATION)
                                    == PackageManager.PERMISSION_GRANTED) {
            mapa.setMyLocationEnabled(true)
            mapa.getUiSettings().setZoomControlsEnabled(true)
            mapa.getUiSettings().setCompassEnabled(true)
        }
        if (lugares.tamaño() > 0) {
            val p = lugares.elemento(0).posicion
            mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(
                    LatLng(p.latitud, p.longitud), 12F))
        }
        for (n in 0 until lugares.tamaño()) {
            val lugar = lugares.elemento(n)
            val p = lugar.posicion
            if (p != null && p.latitud != 0.0) {
                val iGrande = BitmapFactory.decodeResource(
                    getResources(), lugar.tipoLugar.recurso)
                val icono = Bitmap.createScaledBitmap(iGrande,
                    iGrande.getWidth() / 7, iGrande.getHeight() / 7, false)
                mapa.addMarker(
                    MarkerOptions()
                        .position(LatLng(p.latitud, p.longitud))
                        .title(lugar.nombre).snippet(lugar.direccion)
                        .icon(BitmapDescriptorFactory.fromBitmap(icono)))
            }
        }
    }
}
```

El código utilizado es similar al utilizado en el ejercicio anterior. Una diferencia está en que el centro del mapa se sitúa (`moveCamera`) en el primer lugar de `listaLugares` siempre que tenga algún elemento. Luego se introduce un bucle donde añadiremos un marcador para cada lugar. Queremos utilizar los iconos utilizados en la aplicación. El problema es que su tamaño es excesivo. Para resolverlo los leemos como `Drawables` de los recursos, los convertimos en `Bitmap` y los escalamos dividiendo su anchura y altura entre siete.

NOTA: *Desde un punto de vista de eficiencia, lo ideal sería añadir los nuevos recursos reescalados al tamaño final o usar gráficos vectoriales.*

5. Registra esta actividad añadiendo la siguiente línea en *AndroidManifest.xml* dentro de la etiqueta `<application>`:

```
<activity android:name=".presentacion.MapaActivity" />
```

6. Vamos a añadir en la actividad principal una nueva opción en el *ActionBar* para visualizar el mapa. Para ello edita el fichero *res/menu/menu_main.xml* y añade el siguiente ítem de menú:

```
<item android:title="Mapa"
      android:id="@+id/menu_mapa"
      android:icon="@android:drawable/ic_menu_myplaces"
      android:orderInCategory="100"
      app:showAsAction="always"/>
```

7. Abre la clase `MainActivity` y añade dentro del método `onOptionsItemSelected()` el siguiente código:

```
if (id==R.id.menu_mapa) {
   Intent intent = new Intent(this, MapaActivity.class);
   startActivity(intent);
}
```

```
R.id.menu_mapa -> {
   startActivity(Intent(this, MapaActivity::class.java))
   true;
}
```

8. Ejecuta la aplicación en un dispositivo real y selecciona la opción que acabas de introducir. El resultado ha de ser similar al siguiente:



NOTA: Si aparece el error: NoClassDefFoundError: Lorg/apache/ http/ ProtocolVersion consulta [36].

9. Si cambias de orientación el terminal, la actividad Mapa se reinicializará y el mapa volverá a la posición inicial. Si quieres evitarlo, bloquea la orientación de esta actividad. Para ello añade el siguiente atributo en la definición de la actividad dentro de *AndroidManifest.xml*.

```
<activity android:name=".MapaActivity"
       android:screenOrientation="portrait"/>
```

[36] https://stackoverflow.com/questions/50782806



Ejercicio: *Añadiendo un escuchador en Google Maps*

Si en el ejercicio anterior pulsas sobre un marcador, verás como se abre una ventana de información (`InfoWindow`). Queremos conseguir que cuando se pulse sobre esta ventana se abra la actividad que nos muestra información detallada.



1. Vamos a introducir un escuchador que recoja el evento correspondiente cuando se pulse sobre esta ventana. Para ello, añade al final del método `onMapReady()` de la actividad `MapaActivity` la siguiente línea:

```
mapa.setOnInfoWindowClickListener(this);
```

2. Aparecerá un error justo en la línea que acabas de introducir. Si sitúas el cursor de texto sobre el error, aparecerá una bombilla roja con opciones para resolver el error. Selecciona *"Make 'MapaActivity' implement 'OnInfoWindowClickListener'"*. Observa como en la definición de la clase se añade esta interfaz:

```
public class MapaActivity extends FragmentActivity implements
                     OnMapReadyCallback, GoogleMap.OnInfoWindowClickListener {
```

```
class MapaActivity: FragmentActivity(), OnMapReadyCallback,
                                    GoogleMap.OnInfoWindowClickListener {
```

3. Ahora aparecerá un nuevo error sobre `MapaActivity` dado que no implementa esta interfaz. Aparecerá la bombilla roja con opciones para resolver el error. Selecciona *"Implemented methods"* y luego el único método de la interfaz. Completa este código, tal y como se muestra a continuación:

```
@Override public void onInfoWindowClick(Marker marker) {
    for (int pos=0; pos<Lugares.tamaño(); pos++){
        if (Lugares.elemento(pos).getNombre()
                .equals(marker.getTitle())){
            Intent intent = new Intent(this, VistaLugarActivity.class);
            intent.putExtra("pos", pos);
            startActivity(intent);
            break;
        }
    }
}
```

```
override fun onInfoWindowClick(marker: Marker) {
   for (pos in 0 until lugares.tamaño()) {
      if (lugares.elemento(pos).nombre == marker.title) {
         val intent = Intent(this, VistaLugarActivity::class.java)
         intent.putExtra("pos", pos)
         startActivity(intent)
```

```
            break
        }
    }
}
```

Se llamará a este método cuando se pulse sobre cualquier ventana de información. Para averiguar el marcador al que corresponde, se pasa el objeto `Marker` que se ha pulsado. En el marcador hemos introducido alguna información sobre el lugar (como el nombre); sin embargo, lo que necesitamos es el `id` del lugar. No resulta sencillo introducir este `id` en un objeto `Marker`. Para resolverlo hemos introducido un bucle donde se busca un lugar cuyo nombre coincida con el título de marcador. Cuando se encuentre una coincidencia, se creará una intención para lanzar la actividad correspondiente.

NOTA: *Lo más correcto para resolverlo sería crear un descendiente de* `Marker` *que añada este* `id`. *Sin embargo, la clase* `Marker` *se ha marcado como* `final`, *por lo que no es posible crear descendientes.*



Preguntas de repaso: *Google Maps*

CAPÍTULO 8.
Servicios, notificaciones y receptores de anuncios

Las aplicaciones que hemos creado hasta el momento estaban formadas por una serie de actividades, cada una de las cuales permitía construir un elemento de interacción con el usuario. Una aplicación en Android dispone de otros tipos de componentes, que se estudiarán en este capítulo. Cuando necesites que parte de una aplicación se ejecute en segundo plano, debajo de otras actividades, y sin que precise de ningún tipo de interacción con el usuario, la opción más adecuada es crear un servicio. Un servicio puede estar en ejecución indefinidamente, o podemos controlarlo desde una actividad. A lo largo de este capítulo aprenderemos las facilidades proporcionadas para la creación de servicios.

Por otra parte, las notificaciones de la barra de estado constituyen un mecanismo de comunicación vital en Android. Permiten que las aplicaciones que corren en un segundo plano adviertan al usuario sobre alertas, avisos o cualquier tipo de información. Las notificaciones se representan como pequeños iconos en la barra superior de la pantalla y se utilizan habitualmente para indicar la llegada de un mensaje, una cita de calendario, una llamada perdida o cualquier otra incidencia de interés para el usuario. Se trata de una comunicación que no requiere una interacción inmediata del usuario; este puede estar utilizando otra aplicación sin ser interrumpido o puede no estar utilizando el teléfono en ese momento. Este hecho hace de las notificaciones un mecanismo de comunicación ideal para un servicio (o receptores de anuncios, como veremos a continuación). Por lo tanto, este capítulo parece el sitio ideal para describir cómo podemos crear nuestras propias notificaciones y utilizarlas desde nuestras aplicaciones.

Terminaremos el capítulo estudiando otro componente de una aplicación Android: los receptores de anuncios. Un receptor de anuncios (*Broadcast Receiver*, en inglés) permite realizar acciones cuando se producen anuncios globales de tipo *broadcast*. Existen muchos anuncios originados por el sistema (por ejemplo, *Batería baja, Llamada entrante*, etc.). Aunque las aplicaciones también pueden lanzar un anuncio *broadcast* o incluso crear nuevos tipos. Los receptores de anuncios te permitirán crear aplicaciones mucho más integradas en el entorno donde se ejecutan.



Objetivos:

- Describir el uso de servicios en Android.
- Enumerar los pasos a seguir cuando queramos crear un servicio para que una tarea se ejecute en segundo plano.
- Mostrar cómo pueden ser utilizadas las notificaciones de la barra de estado como mecanismo de comunicación eficaz con el usuario.
- Enumerar los pasos a seguir para crear un receptor de anuncios.
- Enumerar los receptores de anuncios más importantes disponibles en Android.
- Describir el uso de receptores de anuncios como mecanismo de comunicación entre aplicaciones.
- Describir el uso de un servicio como mecanismo de comunicación entre aplicaciones.

8.1. Introducción a los servicios en Android



Vídeo[tutorial]: *Los servicios en Android*



Vídeo[tutorial]: *Un servicio para ejecución en segundo plano*

En muchos casos, será necesario añadir un nuevo componente a tu aplicación para ejecutar algún tipo de acción que se ejecute en segundo plano, es decir, que no requiera una interacción directa con el usuario, pero que queramos que permanezca activo aunque el usuario cambie de actividad. Este es el momento de crear un servicio.

En Android los servicios tienen una doble función:

- La primera función permite indicar al sistema que el elemento que estamos creando ha de ejecutarse en segundo plano, normalmente durante un largo período de tiempo. Este tipo de servicios se inician mediante el método `startService()`, que indica al sistema que los ejecute de forma indefinida hasta que alguien le indique lo contrario.
- Los servicios también permiten que nuestra aplicación se comunique con otras aplicaciones, para lo cual ofreceremos ciertas funciones que podrán llamarse desde otras aplicaciones. Este tipo de servicios se inician mediante el método `bindService()`, que permite establecer una conexión con el servicio e invocar alguno de los métodos que ofrece.

Cada vez que se crea un servicio usando `startService()` o `bindService()`, el sistema instancia el servicio y llama al método `onCreate()`. Corresponde al servicio implementar el comportamiento adecuado; habitualmente creará un hilo de ejecución (*thread*) secundario donde se realizará el trabajo.

Un servicio en sí puede ser algo muy simple. En este capítulo se verán ejemplos de servicios locales escritos en muy pocas líneas. No obstante, también pueden complicarse, como veremos al final del capítulo, cuando tratemos de invocar servicios remotos por medio de una interfaz AIDL (*Android Interface Definition Language*).

Un servicio, como el resto de los componentes de una aplicación, se ejecuta en el hilo principal del proceso de la aplicación. Por lo tanto, si el servicio necesita un uso intensivo de CPU o puede quedar bloqueado en ciertas operaciones, como el uso de redes, debes crear un hilo diferente para ejecutar estas acciones. También puedes utilizar la clase `IntentService` para lanzar un servicio en su propio hilo.

8.1.1. Ciclo de vida de un servicio

Es importante que recuerdes que un servicio tiene un ciclo de vida diferente del de una actividad. A continuación podemos ver un gráfico que ilustra su ciclo de vida:

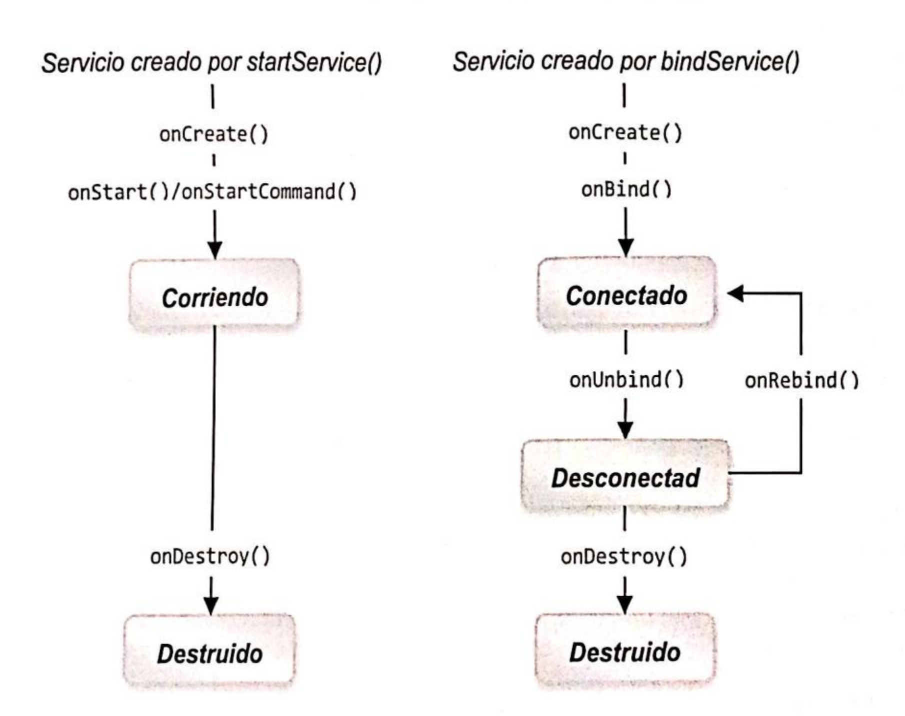


Figura 5: *Ciclo de vida de los servicios.*

Como acabamos de explicar, existen dos tipos de servicios en función de cómo hayan sido creados. Las funciones de estos servicios son diferentes, y por lo tanto, también su ciclo de vida.

Si el servicio se inicia mediante `startService()`, el sistema comenzará creándolo y llamando a su método `onCreate()`. A continuación llamará a su método

`onStartCommand(Intent intent, int flags, int startId)`[37]. El servicio continuará en ejecución hasta que sea invocado el método `stopService()` o `stopSelf()`.

> *NOTA: Si se producen varias llamadas a `startService()`, eso no supondrá la creación de varios servicios, aunque sí que se realizarán múltiples llamadas a `onStartCommand()`. No importa cuántas veces haya sido creado el servicio, parará con la primera invocación de `stopService()` o `stopSelf()`. Sin embargo, podemos utilizar el método `stopSelf(int startId)` para asegurarnos de que el servicio no parará hasta que todas las llamadas hayan sido procesadas.*

Cuando se inicia un servicio para realizar alguna tarea en segundo plano, el proceso donde se ejecuta podría ser eliminado ante una situación de baja memoria. Podemos configurar la forma en que el sistema reaccionará ante esta circunstancia según el valor que devolvamos en `onStartCommand()`. Existen dos modos principales: devolveremos `START_STICKY` si queremos que el sistema trate de crear de nuevo el servicio cuando disponga de memoria suficiente; o devolveremos `START_NOT_STICKY` si queremos que el servicio sea creado de nuevo solo cuando llegue una nueva solicitud de creación.

Conviene aclarar que en situaciones donde el sistema necesite memoria, podrá matar el proceso que contiene nuestro servicio. A la hora de elegir el proceso a eliminar, un servicio es considerado menos prioritario que las actividades visibles, aunque más prioritario que otras actividades en segundo plano. Dado que el número de actividades visibles es siempre reducido, un servicio solo será eliminado en situaciones de extrema necesidad de memoria. Por otra parte, si un cliente visible está conectado a un servicio, el servicio también será considerado como visible, siendo tan prioritario como el cliente. En el caso de un proceso que contenga varios componentes (por ejemplo, una actividad y un servicio), su prioridad se obtiene como el máximo de sus componentes.

También podemos utilizar `bindService(Intent servicio, ServiceConnection conexion, int flags)` para obtener una conexión persistente con un servicio. Si dicho servicio no está en ejecución, será creado (siempre que el *flag* `BIND_AUTO_CREATE` esté activo), llamándose al método `onCreate()`, pero no se llamará a `onStartCommand()`. En su lugar se llamará al método `onBind(Intent intencion)`, que ha de devolver al cliente un objeto `IBinder` a través del cual se podrá establecer una comunicación entre cliente y servicio. Esta comunicación se establece por medio de una interfaz escrita en AIDL, que permite el intercambio de objetos entre aplicaciones que corren en procesos separados. El servicio permanecerá en ejecución tanto tiempo como la conexión esté establecida, independientemente de que se mantenga o no la referencia al objeto `IBinder`.

También es posible diseñar un servicio que pueda ser arrancado de ambas formas (`startService()` y `bindService()`). Este servicio permanecerá activo si ha sido creado desde la aplicación que lo contiene o si recibe conexiones desde otras aplicaciones.

---

[37] En versiones de la API inferiores a 2.0, el método llamado será `onStart()`. En versiones recientes se mantiene por razones de compatibilidad.

8.1.2. Permisos

Podemos conseguir acceso global a un servicio declarado en la etiqueta `<service>` de `AndroidManifest.xml`. También podemos definir un permiso para restringir su acceso. En este caso, las aplicaciones han de declarar este permiso, con el correspondiente `<uses-permission>` en su propio manifiesto.

Podemos definir un permiso para arrancar, parar o conectarse a un servicio. De forma adicional, podemos restringir el acceso a funciones específicas de las ofertadas por un servicio. Para este propósito, podemos llamar al principio de nuestra función a `checkCallingPermission(String)` para verificar si el cliente dispone de un permiso en concreto. Para más información sobre permisos, se recomienda la lectura del capítulo 7.

8.2. Un servicio para ejecución en segundo plano

Dentro de los dos usos de un servicio, el más frecuente es permitirnos ejecutar parte de nuestra aplicación en segundo plano.



Ejercicio: *Un servicio para ejecución en segundo plano de reproducción de música*

Veamos un ejemplo de servicio que corre en el mismo proceso de la aplicación que lo utiliza. El servicio será creado con la finalidad de reproducir una música de fondo y podrá ser arrancado y detenido desde la actividad principal.

1. Crea un nuevo proyecto con nombre *Servicio Música* y tipo *Empty Activity*.
2. Reemplaza el código del *layout* `activity_main.xml` por:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView android:layout_width="match_parent"
             android:layout_height="wrap_content"
             android:text="Servicio de reproducción de música"/>
   <Button android:id="@+id/boton_arrancar"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="Arrancar servicio"/>
   <Button android:id="@+id/boton_detener"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:text="Detener servicio"/>
</LinearLayout>
```

Se trata de un *layout* muy sencillo, con un texto y dos botones:

Servicio de reproducción de música

Arrancar servicio

Detener servicio

3. Reemplaza el código de la actividad por:

```
public class MainActivity extends Activity {
   @Override   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      Button arrancar = findViewById(R.id.boton_arrancar);
      arrancar.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            startService(new Intent(MainActivity.this,
                  ServicioMusica.class));
         }
      });
      Button detener = findViewById(R.id.boton_detener);
      detener.setOnClickListener(new View.OnClickListener() {
         public void onClick(View view) {
            stopService(new Intent(MainActivity.this,
                  ServicioMusica.class));
         }
      });
   }
}
```

4. Crea la nueva clase, `ServicioMusica`, con el siguiente código:

```
public class ServicioMusica extends Service {
   MediaPlayer reproductor;

   @Override public void onCreate() {
      Toast.makeText(this,"Servicio creado",
                              Toast.LENGTH_SHORT).show();
      reproductor = MediaPlayer.create(this, R.raw.audio);
   }

   @Override
   public int onStartCommand(Intent intenc, int flags, int idArranque) {
      Toast.makeText(this,"Servicio arrancado "+ idArranque,
                              Toast.LENGTH_SHORT).show();
      reproductor.start();
      return START_STICKY;
   }
```

```
  @Override public void onDestroy() {
     Toast.makeText(this,"Servicio detenido",
                              Toast.LENGTH_SHORT).show();
     reproductor.stop();
     reproductor.release();
  }

  @Override public IBinder onBind(Intent intencion) {
     return null;
  }
}
```

5. Edita el fichero `AndroidManifest.xml` y añade la siguiente línea dentro de la etiqueta `<application>`:

```
<service android:name=".ServicioMusica" />
```

6. Crea una nueva carpeta que se llame *raw* dentro de la carpeta *res*. Arrastra a su interior el fichero *audio.mp3*.

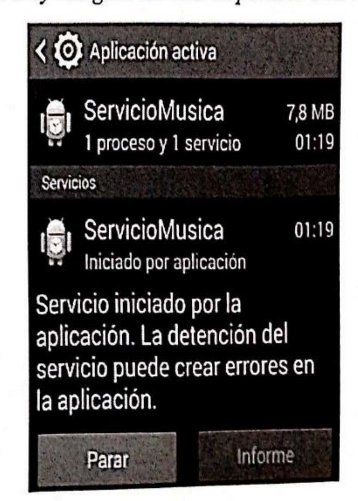
   ***NOTA:** Puedes usar cualquier fichero de música compatible, siempre que el nombre sea* audio. *Ej. http://www.androidcurso.com/images/dcomg/ficheros/audio.mid*

7. Ejecuta la aplicación y comprueba su funcionamiento. Puedes terminar la actividad pulsando la tecla "retorno" y verificar que el servicio continúa en marcha.

8. Verifica que, aunque pulses varias veces el botón *Arrancar servicio*, este no vuelve a crearse, pero sí que vuelve a llamarse al método `onStartCommand()`. Además, con solo una vez que pulses en *Detener servicio*, este parará.

   Ejecuta la aplicación y pon en funcionamiento el servicio. En un dispositivo con versión 6.0 o superior, asegúrate que estén activadas las opciones de desarrollador. Accede a *Ajustes > Opciones del desarrollador > Servicios en ejecución.* En una versión anterior a la 6.0 accede a *Ajustes > Más > Administrador de aplicaciones.* Selecciona la pestaña *EN EJECUCIÓN* y busca *ServicioMusica*. Desde aquí puedes obtener información y detener el servicio.

   ***NOTA:** En versiones muy antiguas no está disponible esta información.*



8.2.1. El método onStartCommand()

El método `onStartCommand()` aparece a partir del nivel de API 5, en sustitución de `onStart()`. Se llama cada vez que un cliente inicializa un servicio mediante el método `startService()`. Veamos con más detalle cómo pueden ser utilizados sus parámetros para obtener información valiosa:

```
public int onStartCommand(Intent intencion, int flags, int idArranque)
```

Los parámetros se detallan a continuación:

`intencion` Un objeto `Intent` que se indicó en la llamada `startService(Intent)`.

`flags` Información adicional sobre cómo arrancar el servico. Puede ser 0, `START_FLAG_REDELIVERY` o `START_FLAG_RETRY`. Un valor distinto de 0 se utiliza para reiniciar un servicio tras detectar algún problema.

`idArranque` Un entero único que representa la solicitud de arranque específica. Usar este mismo entero en el método `stopSelfResult(int idArranque)`.

retorna Describe cómo ha de comportarse el sistema cuando el proceso del servicio sea matado una vez que el servicio ya se ha inicializado. Esto puede ocurrir en situaciones de baja memoria. Los siguientes valores están permitidos:

`START_STICKY`: Cuando sea posible, el sistema tratará de recrear el servicio. Se realizará una llamada a `onStartCommand()`, pero con el parámetro `intencion` igual a `null`. Esto tiene sentido cuando el servicio puede arrancar sin información adicional como, por ejemplo, el servicio mostrado para la reproducción de música de fondo.

`START_NOT_STICKY`: El sistema no tratará de volver a crear el servicio; por lo tanto, el parámetro `intencion` nunca podrá ser igual a `null`. Esto tiene sentido cuando el servicio no puede reanudarse una vez interrumpido.

`START_REDELIVER_INTENT`: El sistema tratará de volver a crear el servicio. El parámetro `intencion` será el que se utilizó en la última llamada `startService(Intent)`.

`START_STICKY_COMPATIBILITY`: Versión compatible de `START_STICKY`, que no garantiza que `onStartCommand()` sea llamado después de que el proceso sea matado.

Preguntas de repaso: *Servicios*

8.3. Un servicio en un nuevo hilo con IntentService

NOTA: La clase `IntentService` *ha sido declarada como obsoleta. En su lugar se recomienda utilizar* `JobIntentService`.

A la hora de diseñar aplicaciones en Android hay que tener muy en cuenta que todos los componentes (actividades, servicios y receptores de anuncios) se van a ejecutar en el hilo principal de la aplicación. Dado que este hilo ha de estar siempre disponible para atender a los eventos generados por el usuario, nunca debe ser bloqueado. Es decir, cualquier proceso que requiera un tiempo importante no ha de ser ejecutado desde este hilo. En su lugar hay que crear un nuevo hilo para que realice este proceso y así dejar libre al hilo principal para que este pueda seguir procesando nuevos eventos.

Podemos crear un nuevo hilo utilizando la clase estándar de Java `Thread`, tal y como se ha explicado en el capítulo 5. Para automatizar este proceso, Android nos proporciona la clase `AsyncTask`. También nos proporciona la clase `IntentService`, cuando queramos lanzar un servicio en un nuevo hilo. En este apartado veremos qué ocurre cuando un servicio bloquea el hilo principal y cómo solucionarlo mediante la clase `IntentService`.



Ejercicio: *Un servicio que bloquea el hilo principal*

Muchos servicios han de realizar costosas operaciones o han de esperar a que concluyan lentas operaciones en la red. En ambos casos hay que tener la precaución de no bloquear el hilo principal. De hacerlo, el resultado puede ser catastrófico, como se muestra en este ejercicio.

1. Crea un nuevo proyecto con nombre *IntentService*, de tipo *Empty Activity* y cuyo nombre de paquete sea `com.example.intentservice`.
2. Reemplaza el código del *layout* principal por:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical" >
   <LinearLayout
      android:layout_width="match_parent"
      android:layout_height="wrap_content" >
      <EditText
         android:id="@+id/entrada"
         android:layout_width="0dip"
         android:layout_height="wrap_content"
         android:layout_weight="1"
         android:inputType="numberDecimal"
         android:text="2.2" >
         <requestFocus />
```

```
        </EditText>
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:onClick="calcularOperacion"
            android:text="Calcular operación" />
    </LinearLayout>

    <TextView
        android:id="@+id/salida"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text=" "
        android:textAppearance="?android:attr/textAppearanceMedium"/>
</LinearLayout>
```

3. Reemplaza el código de `MainActivity` por el siguiente:

```
public class MainActivity extends Activity {
    private EditText entrada;
    public static TextView salida;

    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        entrada = findViewById(R.id.entrada);
        salida = findViewById(R.id.salida);
    }

    public void calcularOperacion(View view) {
        double n = Double.parseDouble(entrada.getText().toString());
        salida.append(n +"^2 = ");
        Intent i = new Intent(this, ServicioOperacion.class);
        i.putExtra("numero", n);
        startService(i);
    }
}
```

Observa que la variable `salida` ha sido declarada como **`public static`**. Esto nos permitirá acceder a esta variable desde otras clases. Se llamará al método `calcularOperacion()` cuando se pulse el botón. Comienza obteniendo el valor real introducido en `entrada`. Se muestra la operación a realizar por `salida`. Luego, se crea una nueva intención con nuestro contexto y la clase con el servicio que se define a continuación. Luego se le añade un extra con el valor introducido. Finalmente se arranca el servicio.

4. Crea la clase `ServicioOperacion` con el siguiente código:

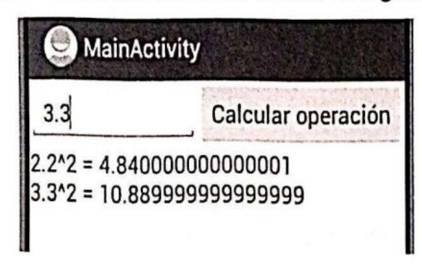
```
public class ServicioOperacion extends Service {
    @Override
    public int onStartCommand(Intent i, int flags, int idArranque){
        double n = i.getExtras().getDouble("numero");
        SystemClock.sleep(5000);
        MainActivity.salida.append(n*n + "\n");
```

```
        return START_NOT_STICKY;
    }

    @Override
    public IBinder onBind(Intent arg0) {
        return null;
    }
}
```

Cuando se arranca el servicio se llamará al método `onStartCommand()`. Este comienza obteniendo el valor a calcular a partir de un extra. Luego vamos a simular que se realizan un gran número de operaciones. Para ello vamos a bloquear el hilo durante 5000 ms (5 segundos) utilizando el método `sleep()`. Una vez terminado, el resultado se muestra directamente en el `TextView salida`. Esta forma de trabajar no resulta muy recomendable. Se ha realizado así para ilustrar como es posible acceder al sistema gráfico de Android dado que estamos en el hilo principal. Finalmente, devolvemos `START_NOT_STICKY` para indicar al sistema que si por fuerza mayor ha de destruir el servicio, no hace falta que lo vuelva a crear.

5. Recuerda registrar el servicio en *AndroidManifest.xml*.
6. Ejecuta la aplicación. El resultado ha de ser similar al siguiente:



Observa como mientras se realiza la operación el usuario no puede pulsar el botón ni modificar el EditText. El usuario tendrá la sensación de que la aplicación está bloqueada.

7. Modifica el tiempo de retardo para que este sea de 25 seg. (`sleep(25000)`). Ejecuta de nuevo la aplicación y observa como el sistema nos mostrará el siguiente error:



8. Para que esto no bloquee el hilo principal podemos utilizar un `IntentService`. En el siguiente ejercicio mostraremos cómo realizarlo.

8.3.1. La clase IntentService

Utilizaremos la clase `IntentService` en lugar de `Service` cuando queramos un servicio que se ejecute en su propio hilo. Esta clase tiene un constructor donde hay que indicar en un `String` el nombre que queremos dar al servicio. Lo habitual será que cuando extendamos esta clase en el constructor llamemos al constructor padre pasándole este nombre. A continuación se muestra un ejemplo de código:

```
public class MiServicio extends IntentService{

   public MiServicio () {
      super("Nombre de mi servicio");
   }

   @Override
   protected void onHandleIntent(Intent intencion) {
      ...
   }
}
```

El siguiente método que hay que sobrescribir es `onHandleIntent`. Este método se lanzará cada vez que se arranque el servicio, pero en este caso se lanzará en hilo nuevo. A través del parámetro *intención* se podrán enviar datos en forma de extras. Es importante destacar que si se lanzan varias peticiones de servicio, estas se pondrán en una cola. Se irán atendiendo una tras otra sin que haya dos a la vez en ejecución. Este comportamiento puede ser interesante para algunas tareas, pero no para otras. Por ejemplo, si tenemos que implementar un servicio de descarga de ficheros, seguramente será más interesante permitir la descarga de varios ficheros en paralelo y no tener que descargarlos de uno en uno.

Finalmente, para lanzar un `IntentService` hay que usar `startService()`. Se realiza exactamente igual que para lanzar un `Service`.



Ejercicio: *Un servicio en su propio hilo*

En este ejercicio aprenderemos a crear servicios que se ejecutan en un hilo de ejecución diferente del principal utilizando la clase `IntentService`. Además, veremos algunas limitaciones de este tipo de servicios, como la imposibilidad de acceder al sistema gráfico.

1. Abre el proyecto IntentService creado en el ejercicio anterior.
2. Crea la clase `IntentServiceOperacion` con el siguiente código:

```
public class IntentServiceOperacion extends IntentService{
   public IntentServiceOperacion() {
      super("IntentServiceOperacion");
   }

   @Override
```

```
    protected void onHandleIntent(Intent intent) {
        double n = intent.getExtras().getDouble("numero");
        SystemClock.sleep(5000);
        MainActivity.salida.append(n*n + "\n");
    }
}
```

3. En `MainActivity` reemplaza la línea:

```
Intent i = new Intent(this, ServicioOperacion.class);
```

por:

```
Intent i = new Intent(this, IntentServiceOperacion.class);
```

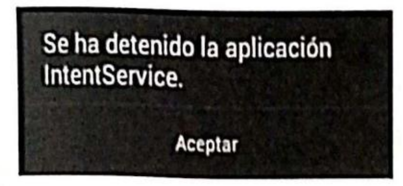
4. En *AndroidManifest.xml* reemplaza la línea:

```
<service android:name=".ServicioOperacion" />
```

por:

```
<service android:name=".IntentServiceOperacion" />
```

5. Ejecuta la aplicación. Tras pulsar el botón el resultado ha de ser:



6. Abre la vista *LogCat* y busca el siguiente Error:

```
FATAL EXCEPTION: IntentService[IntentServiceOperacion]
android.view.ViewRootImpl$CalledFromWrongThreadException: Only th ⏎
e original thread that created a view hierarchy can touch its vie ⏎
ws.
```

Te indica que solo desde el hilo principal se va a poder interactuar con las vistas de la interfaz de usuario. También está prohibido usar la clase `Toast` desde otros hilos.

Como un hilo que hemos creado pertenece al mismo proceso que el hilo principal, compartimos con este todas las variables. Para devolver el valor calculado, podríamos implementar un método o variable públicos, tanto en la clase del servicio como de la actividad. No obstante, vamos a resolver este problema utilizando un mecanismo más elegante, los receptores de anuncios. Se explica en el siguiente apartado.



Preguntas de repaso: *Servicios e hilos*



8.4. Las notificaciones de la barra de estado



Vídeo[tutorial]: *Notificaciones en Android*

La barra de estado de Android se encuentra situada en la parte superior de la pantalla. La parte izquierda de esta barra está reservada para visualizar notificaciones. Cuando se crea una nueva notificación, aparece un pequeño icono que permanecerá en la barra para recordar al usuario la notificación.



El usuario puede arrastrar la barra de notificaciones hacia abajo, para mostrar la lista de las notificaciones por leer. A continuación se muestra un ejemplo:

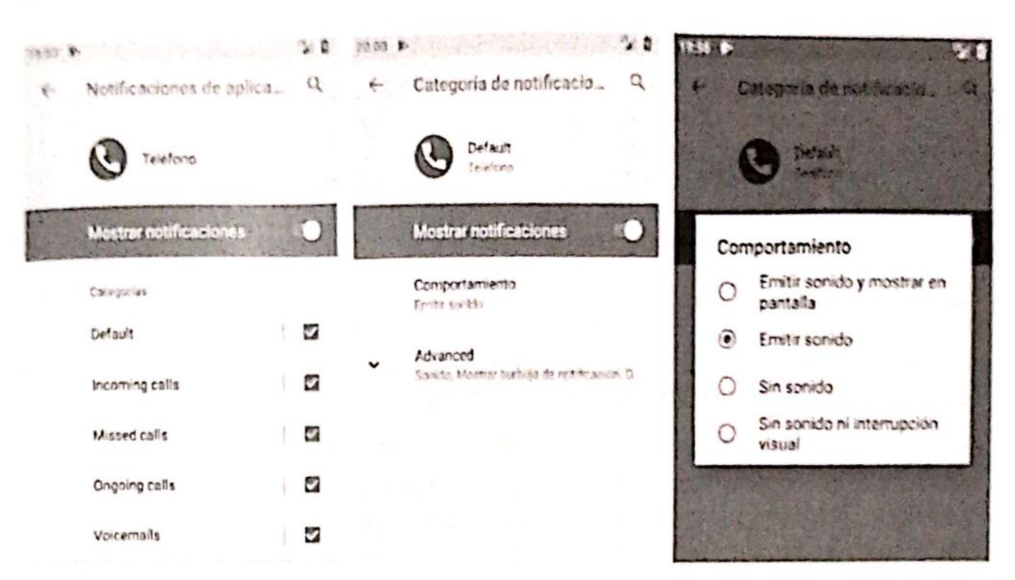


Una notificación puede ser creada por un servicio o por una actividad. Aunque dado que la actividad dispone de su propia interfaz de usuario, parece que las notificaciones son el mecanismo de interacción más interesante del que disponen los servicios. Las notificaciones pueden crearse desde un segundo plano, sin interferir con la actividad que en ese momento esté utilizando el usuario.

Para lanzar una notificación desde Android 8.0 Oreo (API 26) es necesario hacerlo dentro de un canal de notificación. Si se te olvida crear el canal, la notificación no aparecerá. Esto permite a los usuarios tratar todas las notificaciones de un mismo canal de forma conjunta (desactivarla, configurar sonido...). Una misma aplicación puede crear varios canales, uno por cada tipo de notificación. Por ejemplo, en WhatsApp podríamos crear un canal para cada uno de los grupos.

Para comprender mejor este concepto, en un terminal (o emulador) con versión 8 o superior, accede a Ajustes / Aplicaciones y notificaciones / Teléfono / Gestion de Notificaciones de la aplicación. Se mostrarán todos los canales de notificación creados por esta aplicación. Observa como en *Ajustes* a los *canales* se les llama *categorías*:





Desde esta pantalla podrás desactivar cualquier canal de notificaciones. Si pulsas sobre su nombre, podrás configurar el canal. En *Comportamiento* puedes configurar la importancia de una notificación para determinar si el usuario ha de ser interrumpido, tanto visual como acústicamente. Los cuatro posibles niveles se muestran a la derecha. También podemos configurar el sonido asociado o si queremos que aparezcan en modo no molestar.



Ejercicio: *Creación de una notificación*

1. Abre el proyecto ServicioMusica.
2. Asegúrate de que tu proyecto incluye la librería de compatibilidad v7 o superior. Para ello, en *Project/Gradle Script/build.gradle (Module: app)* añade la línea:

```
dependencies {
   …
   implementation 'androidx.appcompat:appcompat:1.2.0'
}
```

3. Declara las siguientes variables al comienzo de la clase `ServicioMusica`:

```
private NotificationManager notificationManager;
static final String CANAL_ID = "mi_canal";
static final int NOTIFICACION_ID = 1;
```

4. Para crear una nueva notificación añade al principio del método `onStartCommand()` las siguientes líneas:

```
notificationManager = (NotificationManager)
                              getSystemService(NOTIFICATION_SERVICE);
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
   NotificationChannel notificationChannel = new NotificationChannel(
         CANAL_ID, "Mis Notificaciones",
         NotificationManager.IMPORTANCE_DEFAULT);
```

```
    notificationChannel.setDescription("Descripcion del canal");
    notificationManager.createNotificationChannel(notificationChannel);
}
NotificationCompat.Builder notificacion =
        new NotificationCompat.Builder(this, CANAL_ID)
                .setSmallIcon(R.mipmap.ic_launcher)
                .setContentTitle("Título")
                .setContentText("Texto de la notificación.");
notificationManager.notify(NOTIFICACION_ID, notificacion.build());
```

Lo primero que necesitamos es una referencia al `NotificationManager` que permite manejar las notificaciones del sistema.

El siguiente paso es crear el canal. Solo puede hacerse en dispositivos con versión 8 o superior; no obstante, es obligatorio que crees el canal, de lo contrario, las notificaciones no se crearán en dispositivos con estas versiones. Para crear el canal has de indicar un ID, un nombre y un nivel de importancia. El nivel de importancia corresponde con los cuatro niveles de comportamiento mostrados en las capturas de *Ajustes*.

La notificación se crea utilizando una clase especial `Builder` que dispone de varios métodos `set` para configurarla. `setContentTitle()` permite indicar el título que describe la notificación y `setContextText()`, información más detallada. Con `setSmallIcon()` indicamos el icono a visualizar. En el ejemplo usamos el mismo que el de la aplicación, aunque no resulta muy adecuado dado que estos iconos han de seguir una estética concreta.

El método `notify()` es el encargado de lanzar la notificación. En el primer parámetro se indica un id para poder identificar esta notificación en un futuro y en el segundo la notificación.

5. Ejecuta la aplicación y verifica el resultado.



Ejercicio: *Lanzar una actividad desde una notificación*

Cuando arrastras la barra de notificaciones hacia abajo, para mostrar la lista de notificaciones, y pulsas sobre una de ellas, es habitual que se abra una actividad para realizar acciones relacionadas con la notificación. En este ejercicio aprenderemos a asociar una actividad a una notificación.

1. Justo después de la creación de la variable `notificacion` introducida en el ejercicio anterior, y antes de la última línea, añade el siguiente código:

```
PendingIntent intencionPendiente = PendingIntent.getActivity(
     this, 0, new Intent(this, MainActivity.class), 0);
notificacion.setContentIntent(intencionPendiente);
```

Este código asocia una actividad que se ejecutará cuando el usuario pulse sobre la notificación. Para ello, se crea un `PendingIntent` (intención pendiente que se ejecutará más adelante) asociado a la actividad `MainActivity`. Por supuesto, también puedes crear una nueva actividad para usarla exclusivamente con este

fin. En un ejemplo más complejo, puedes pasar los parámetros adecuados a través del `Intent`, para que la actividad conozca los detalles específicos que provocaron la notificación (por ejemplo, el número de teléfono que provocó la llamada perdida).

2. Ejecuta la aplicación y verifica el resultado.



Ejercicio: *Eliminar una notificación*

Eliminar una notificación desde código resulta muy sencillo. Se describe en este ejercicio:

1. Queremos que si el servicio deja de estar activo, elimine la notificación. Para ello añade en `onDestroy()`:

```
notificationManager.cancel(NOTIFICACION_ID);
```

Este paso es opcional. Muchas notificaciones han de permanecer visibles aunque el servicio que las creó sea destruido. En nuestro caso, dado que estamos anunciando que un servicio de reproducción de música está activado, la notificación deja de tener sentido al desaparecer el servicio.

2. Ejecuta la aplicación y verifica que al parar el servicio la notificación desaparece.



Práctica: *Uso del servicio de música en Asteroides*

1. Copia la clase `ServicioMusica` del ejercicio anterior en el proyecto Asteroides.
2. Corrige los errores que hayan aparecido para adaptarla al nuevo proyecto.
3. En el ejercicio anterior, cuando se visualizaban los detalles de la notificación se podía lanzar la actividad `MainActivity` del proyecto *ServicioMusica*. Ahora ha de lanzarse la actividad `MainActivity` del proyecto *Asteroides*.
4. Si realizas el punto anterior simplemente lanzando la actividad `MainActivity`, cuando el usuario pulse sobre la notificación el sistema lanzará una nueva tarea, aunque ya exista una previa. Si te interesa que no se lance una nueva tarea cuando ya exista una previa, añade la línea en negrita en *AndroidManifest.xml*.

```
<activity android:name=".MainActivity"
          android:label="@string/app_name"
          android:launchMode="singleTask">
```

5. Lanza el servicio en el método `onCreate()` de la actividad `MainActivity`. Para el servicio en el método `onDestroy()`.





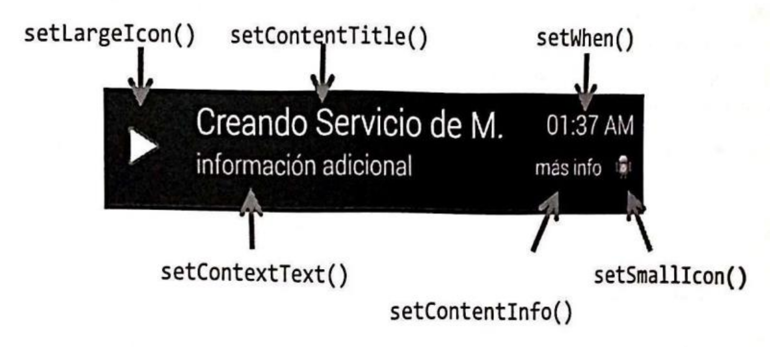
Ejercicio: *Características avanzadas con notificaciones*

Lo aprendido hasta ahora es suficiente para cubrir la mayoría de las notificaciones. No obstante, en este ejercicio aprenderemos a configurar otras características que en ciertos casos podrían ser interesantes.

1. En el proyecto `ServicioMusica` añade las siguientes líneas en la creación del objeto `NotificationCompat.Builder`:

```
.setLargeIcon(BitmapFactory.decodeResource(getResources(),
                                android.R.drawable.ic_media_play))
.setWhen(System.currentTimeMillis() + 1000 * 60 * 60)
.setContentInfo("más info")
.setTicker("Texto en barra de estado")
```

   El siguiente esquema muestra dónde se visualiza parte de la información que acabamos de introducir:



   El método `setLargeIcon()` permite visualizar un icono de mayor resolución que se muestra a la izquierda de la notificación. `setWhen()` permite indicar la hora en que ocurrió el evento. Solo actúa a efectos de visualización. Aunque se indique una hora futura, la notificación se lanzará inmediatamente. Las notificaciones en el panel se ordenan por este tiempo. `setContentInfo()` muestra un texto ubicado en la parte inferior derecha de la notificación. Finalmente `setTicker()` muestra un texto en la barra de estado, durante unos segundos, cuando la notificación se crea por primera vez.

2. Ejecuta la aplicación y verifica el resultado.

8.4.1. Configurando tipos de avisos en las notificaciones

Como hemos comentado, una notificación puede utilizar diferentes métodos para alertar al usuario de que se ha producido. Veamos algunas opciones.

Asociar un sonido

Si consideras que una notificación es muy urgente y deseas que el usuario pueda conocerla de forma inmediata, puedes asociarle un sonido, que se reproducirá cuando se produzca la notificación.

El usuario puede definir un sonido por defecto para las notificaciones. Si quieres asociar el sonido de notificaciones por defecto, utiliza la siguiente sentencia en la creación del objeto `NotificationCompat.Builder`:

```
.setDefaults(Notification.DEFAULT_SOUND)
```

Si prefieres reproducir un sonido personalizado para la notificación, puedes copiar un fichero de audio en la carpeta res/raw del proyecto (por ejemplo, el fichero explosión.mp3) y añadir la siguiente sentencia:

```
.setSound(Uri.parse("android.resource://" + getPackageName() + "/"
                                        + R.raw.explosion));
```

Añadiendo vibración

También es posible alertar al usuario haciendo vibrar el teléfono. Puedes utilizar la vibración por defecto:

```
.setDefaults(Notification.DEFAULT_VIBRATE);
```

O, por el contrario, tu propio patrón de vibración:

```
.setVibrate(new long[] { 0,100,200,300 })
```

El *array* define un patrón de longitudes expresadas en milisegundos, donde el primer valor es el tiempo sin vibrar; el segundo es el tiempo vibrado; el tercero, el tiempo sin vibrar, y así sucesivamente. Este *array* puede ser tan largo como queramos, pero solo se activará una vez, no se repetirá de forma cíclica.

Añadiendo parpadeo de LED

Algunos móviles disponen de diodos LED que pueden utilizarse para avisar al usuario de que se ha producido una notificación. Este método es muy interesante si el grado de urgencia del aviso no es lo suficientemente alto para usar uno de los métodos anteriores. Podemos utilizar el aviso de LED configurado por defecto:

```
.setDefaults(Notification.DEFAULT_LIGHTS)
```

O podemos definir una cadencia de tiempo y color específica para nuestra notificación:

```
.setLights(Color.RED, 3000, 1000);
```

En el ejemplo anterior indicamos que queremos que el LED se ilumine en color rojo durante 3000 ms y luego esté apagado durante 1000 ms. Esta secuencia se repetirá de forma cíclica hasta que el usuario atienda la notificación.

Conviene destacar que no todos los móviles disponen de un LED para este propósito, y algunos dispositivos con LED no soportan notificaciones con parpadeo.

Además, no todos los colores pueden ser utilizados. Otro aspecto a tener en cuenta es que el sistema solo activa el LED cuando la pantalla está apagada.



Práctica: *Una notificación de socorro*

1. En el proyecto anterior, crea un nuevo botón.
2. Al pulsar este botón se lanzará una nueva notificación que mostrará el texto «¡SOCORRO!».
3. El audio de la notificación será una grabación de voz que diga «¡SOCORRO!».
4. La notificación hará vibrar el teléfono con el mensaje internacional de socorro S.O.S. codificado en Morse. Para ello, haz vibrar el teléfono con una sucesión de tres pulsaciones cortas, tres largas y otras tres cortas (. . . - - - . . .).

Desde Android 8 resulta más interesante configurar estas características especiales en un canal de notificación, en lugar de hacerlo en cada notificación. Para ello no tienes más que añadir las líneas subrayadas, cuando crees el canal:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
   NotificationChannel notificationChannel = new NotificationChannel(
         CANAL_ID, "Mis Notificaciones",
         NotificationManager.IMPORTANCE_DEFAULT);
   notificationChannel.setDescription("Descripcion del canal");
   notificationChannel.enableLights(true);
   notificationChannel.setLightColor(Color.RED);
   notificationChannel.setVibrationPattern(new long[]{0, 100, 300, 100});
   notificationChannel.enableVibration(true);
   notificationManager.createNotificationChannel(notificationChannel);
}
```



Preguntas de repaso: *Notificaciones*

8.4.2. Servicios en primer plano

Hasta en Android 8 (API 26) una aplicación podría ejecutarse libremente en segundo plano. A partir de esta versión se va a limitar la ejecución en segundo plano. Los motivos que ha llevado a Google a tomar esta decisión es que las aplicaciones en segundo plano son grandes consumidoras de batería. Además, afectan al rendimiento de la aplicación en primer plano, lo que supone una peor experiencia de usuario (especialmente por el consumo de RAM).

Estas restricciones no se aplican si la aplicación se encuentra en primer plano. Se considera que una aplicación se encuentra en primer plano si se cumple alguno de los siguientes puntos:



- Tiene una actividad visible, independientemente de que la actividad se haya iniciado o esté en pausa.
- Otra aplicación en primer plano está conectada a la aplicación, ya sea por vinculación a uno de sus servicios o por el uso de uno de sus proveedores de contenido.
- Tiene un servicio en primer plano.

En concreto, las restricciones a una aplicación que no está en primer plano es:

- No podemos arrancar un servicio.
- Cuando la aplicación pase a segundo plano, los servicios creados permanecen activos varios minutos. Pasado este tiempo, son detenidos.

Si necesitas un servicio que siga funcionando, aunque tu aplicación no esté en primer plano, has de pasarlo a primer plano. Un servicio en primer plano es un servicio que se considera que el usuario es consciente de su actividad. Para conseguir esto, el servicio en primer plano debe proporcionar una notificación para la barra de estado.

Esta notificación es de tipo "en curso", lo que significa que la notificación no se puede descartar, salvo que el servicio se detenga o se quite del primer plano.

Por ejemplo, un reproductor de música que reproduce música desde un servicio se debe configurar para que se ejecute en primer plano, porque el usuario está explícitamente al tanto de su operación. La notificación en la barra de estado puede indicar la canción actual y permitir que el usuario lance una actividad para interactuar con el reproductor de música.

Para solicitar que tu servicio se pase a primer plano, créalo como se ha indicado y llama al siguiente método desde dentro del servicio:

```
startForeground(NOTIFICACION_ID, notificacion);
```

Los dos parámetros son los mismos que usábamos para lanzar una notificación:

```
notificationManager.notify(NOTIFICACION_ID, notificacion);
```

Pero, cuidado, no has de lanzar la notificación, solo has de prepararla y pasárla en el parámetro.

Para quitar un servicio de primer plano utiliza el siguiente método:

```
stopForeground(quitarNotificacion);
```

El parámetro es un valor boleano donde se indica si queremos que la notificación sea retirada.

Desde Android 9, para usar servicios en primer plano, hay que solicitar el permiso `FOREGROUND_SERVICE`. Este es un permiso normal, de modo que el sistema se lo otorga automáticamente, sin que el usuario tenga que dar el visto bueno.

> ***NOTA:*** *A partir de Android 8 no se permite que tu aplicación lance un servicio si está en segundo plano. Sí que podrás lanzarlo en primer plano, pero para ello utiliza el método* `startForegroundService()` *en lugar de* `startService()`.



Ejercicio: *Servicio de música en primer plano*

1. Abre el proyecto Servicio de música y ejecútalo en un dispositivo con una versión de Android 8.0 o superior.

2. Sin salir de la actividad principal, pon otra aplicación en primer plano. La música seguirá escuchándose. Sin embargo, pasado unos 40 segundos, el servicio será detenido, llamando al método `onDestroy()`.

3. Para evitarlo, tenemos que hacer que el servicio sea considerado de primer plano asociándole una notificación. Dentro de `onStartCommand()` crea una notificación tal y como se hizo en el ejercicio *Creación de una notificación*. Personaliza los textos para que sean adecuados a un servicio de música.

4. Queremos que si el usuario pulsa sobre la notificación se abra `MainActivity`. Para ello sigue las instrucciones del ejercicio *Lanzar una actividad desde una notificación*.

5. Elimina la línea donde se lanzaba la notificación y reemplazala por una llamada a `startForeground()`:

```
notificationManager.notify(NOTIFICACION_ID, notificacion.build());
startForeground(NOTIFICACION_ID, notificacion.build());
```

6. En AndroidManifest solicita el permiso `FOREGROUND_SERVICE`.

```
<uses-permission android:name="android.permission.FOREGROUND_SERVICE"/>
```

7. Ejecuta la aplicación. Comprueba como aparece una notificación que indica que el servicio está en marcha.

8. Si muestras los detalles de las notificaciones y tratas de descartarla, verás que no es posible.

Servicio Música 2:15 p. m.
Servicio de música
Disfruta de tu música preferida

9. Pasa la aplicación a segundo plano. Verifica que el servicio no es detenido.

10. Pulsa sobre la notificación, se abrirá la actividad con los dos botones. Desde aquí sí que podrás detener el servicio.

8.5. Receptores de anuncios

Un receptor de anuncios (`BroadcastReceiver`) recibe anuncios globales de tipo *broadcast* y reacciona ante ellos. Existen muchos originados por el sistema como, por ejemplo, *Batería baja* o *Llamada entrante* (más adelante se muestra una tabla). Aunque las aplicaciones también pueden lanzar sus propios anuncios *broadcast*.

Un anuncio *broadcast* se envía y se recibe en forma de intención, donde se describe el evento o la acción que ha ocurrido y se puede adjuntar información adicional.

Los receptores de anuncios no tienen interfaz de usuario, aunque pueden iniciar una actividad o crear una notificación para informar al usuario. El ciclo de vida de un `BroadcastReceiver` es muy sencillo, solo dispone del método `onReceive()`. De hecho, un objeto `BroadcastReceiver` solo existe durante la llamada a `onReceive()`. El sistema crea el `BroadcastReceiver`, llama a este método y cuando termina destruye el objeto.

El método `onReceive()` es ejecutado por el hilo principal de la aplicación. Por lo tanto, no debe bloquear el sistema (véase el ciclo de vida de una actividad). Si tienes que realizar una acción que puede bloquear el sistema, tendrás que lanzar un hilo secundario. Si queremos una acción persistente en el tiempo, resulta muy frecuente lanzar un servicio. Desde un `BroadcastReceiver` no se puede mostrar un cuadro de diálogo o unirse a un servicio (`bindService()`). Para lo primero, en su lugar puedes lanzar una notificación. Para lo segundo, puedes utilizar `startService()` para arrancar un servicio.

Una aplicación puede registrar un receptor de anuncios de dos maneras: en *AndroidManifest.xml* y en tiempo de ejecución mediante el método `registerReceiver()`.

8.5.1. Receptor de anuncios registrado en *AndroidManifest.xml*

Registrar un receptor de anuncios desde *AndroidManifest.xml* es muy sencillo. No tienes más que introducir las siguientes líneas en *AndroidManifest.xml* dentro de la etiqueta `<application>`:

```
<receiver android:name=".ReceptorAnuncio">
   <intent-filter>
      <action android:name="android.intent.action.BATTERY_LOW"/>
   </intent-filter>
</receiver>
```

En segundo lugar tienes que crear la clase `ReceptorAnuncio`. Se llamará al método `onReceive()` cuando el sistema lance el anuncio *broadcast* `BATTERY_LOW`. Esto ocurrirá cuando detecte un nivel bajo de batería.

```
public class ReceptorAnuncio extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //...
  }
}
```

La principal ventaja de declarar un receptor de anuncios en el manifiesto es que en caso de que nuestra aplicación no esté en ejecución, esta será puesta en ejecución de forma automática.

A partir de Android 8 no se pueden registrar receptores de anuncios desde el Manifiesto, a no ser que se encuentren en la lista blanca de anuncios permitidos[38]. Si no están en esta lista, podremos programar nuestro receptor de anuncio por código.



Ejercicio: *Registrar receptor de anuncios en Manifiesto*

1. Crea un nuevo proyecto con nombre *Receptor Anuncio* de tipo *Empty Activity*.
2. En *AndroidManifest.xml*, dentro de `<application>`, añade:

```
<receiver android:name="ReceptorLlamadas" >
   <intent-filter>
     <action android:name="android.intent.action.PHONE_STATE"/>
   </intent-filter>
</receiver>
```

   De esta forma registramos un receptor de anuncios que se activará cuando cambie el estado del teléfono. Este anuncio se encuentra en la lista blanca, lo que nos permite registrarlo en el manifiesto.

3. Tenemos que pedir permiso para leer el estado del teléfono. Añade la siguiente línea dentro de `<manifest>`.

```
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
```

4. Añade también el permiso para conocer el número de teléfono que nos llama.

```
<uses-permission android:name="android.permission.READ_CALL_LOG"/>
```

5. Crea una nueva clase que se llame `ReceptorLlamadas` con el siguiente código:

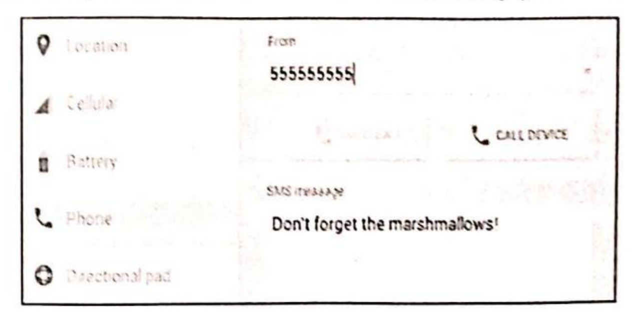
```
public class ReceptorLlamadas extends BroadcastReceiver {

   @Override public void onReceive(Context context, Intent intent) {
      // Sacamos información de la intención
      String estado = "", numero = "";
      Bundle extras = intent.getExtras();
      if (extras != null) {
         estado = extras.getString(TelephonyManager.EXTRA_STATE);
         if (estado.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
            numero = extras.getString(
                         TelephonyManager.EXTRA_INCOMING_NUMBER);
            String info = estado + " " + numero;
            Log.d("ReceptorAnuncio", info + " intent=" + intent);
            Toast.makeText(context, "Llamada: "+ numero,
                  Toast.LENGTH_SHORT).show();
         }
      }
   }
}
```

[38] https://developer.android.com/guide/components/broadcast-exceptions?hl=es-419

6. Ejecuta la aplicación e introduce una llamada. El permiso solicitado es de tipo peligroso, por lo tanto, has de solicitar el permiso al usuario por código. También puedes dar el permiso a la aplicación manualmente desde la configuración.

7. Si utilizas el emulador, puedes utilizar los tres puntos que aparecen en la parte inferior de la barra de herramientas, para abrir los controles extendidos. Selecciona *Phone*, introduce un número de teléfono y pulsa *CALL DEVICE*.



8. Verifica que se muestra el Toast y la información mostrada en el LogCat.

9. Cierra la aplicación o, si lo prefieres, reinicia el teléfono. Sin lanzar la aplicación haz que el teléfono reciba una llamada. Comprueba que el receptor de anuncios se activa igualmente sin haber arrancado la aplicación.

8.5.2. Receptor de anuncios registrado por código

También podemos registrar el anuncio por código, por ejemplo, dentro de una actividad o un servicio. Por ejemplo, podemos crear una clase interna a la actividad de tipo `BroadcastReceiver`.

```
public class MainActivity extends Activity {

  @Override
   protected void onCreate(Bundle savedInstanceState) {
      …
      IntentFilter filtro = new IntentFilter(
                                 "android.intent.action.BATTERY_LOW");
      filtro.addCategory(Intent.CATEGORY_DEFAULT);
      registerReceiver(new ReceptorAnuncio(), filtro);
   }

   public class ReceptorAnuncio extends BroadcastReceiver {
      @Override
      public void onReceive(Context context, Intent intent) {
         …
      }
   }
}
```

La principal ventaja de declarar un receptor de anuncios dentro de otra clase es que podemos acceder a las variables y métodos de la clase que nos contiene.

Un inconveniente obvio es que el receptor no estará activo hasta que comience la actividad o servicio que lo programa.

A partir de Android 8 hay que registrar receptores de anuncios desde código, con la excepción de los que se encuentren en la lista blanca de anuncios permitidos. Si necesitas un receptor de anuncios activo, aunque el usuario no utilice tu aplicación, siempre puedes registrarlo desde un servicio en primer plano.



Ejercicio: *Registrar receptor de anuncios por código*

1. Crea un nuevo proyecto o utiliza el anterior.
2. En el layout de la actividad añade un id al `TextView` que muestra `"Hello World!"` con valor `textView`.
3. Añade la siguiente variable a `MainActivity`:

```
private TextView textView;
```

4. Añade al final de `onCreate()`:

```
textView = findViewById(R.id.textView);
IntentFilter filtro = new IntentFilter(Intent.ACTION_POWER_DISCONNECTED);
filtro.addCategory(Intent.CATEGORY_DEFAULT);
registerReceiver(new ReceptorAnuncios(), filtro);
```

   Empezamos inicializando `textView` desde el layout. Las siguientes tres líneas permiten registrar un receptor por código. En concreto el que corresponde al anuncio bradacast `"android.intent.action.ACTION_POWER_DISCONNECTED"` que es el valor de la constante `ACTION_POWER_DISCONNECTED`. Este anuncio es enviado por el sistema cuando el usuario deconecta el móvil de la alimentación.

5. Añade la siguiente clase, dentro de `MainActivity`, justo antes de la última `}`.

```
public class ReceptorAnuncios extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      textView.setText(textView.getText()+" DESCONECTADO");
   }
}
```

   Cada vez que llegue un anuncio se llamará a `onReceive()`. Este método pertenece a la clase `ReceptorAnuncios`, pero también está dentro de `MainActivity`, por lo tanto, puede acceder a sus variables. Lo que se va a hacer es añadir a `textView` la cadena `" DESCONECTADO"`.

6. Ejecuta la aplicación. En un móvil desconecta el cable USB. En un emulador utiliza los tres puntos de herramientas y pulsa en *Battery*. En *Chager Connection* selecciona *None*. Obserba como en la actividad se muestra `"DESCONECTADO"`. Puedes repetir la operación varias veces.



Recursos adicionales: *Lista de anuncios* broadcast

La siguiente lista muestra los anuncios *broadcast* más importantes organizados por temas. La columna de la izquierda muestra la constante en Java que identifica la acción *broadcast*. En la columna de la derecha, (*No Manifest)*: indica que no se puede declarar el receptor de anuncios en *AndroidManifest.xml*. Solo se puede utilizar `registerReceiver()`. *(Solo sistema)*: indica que se trata de una intención protegida, que solo puede ser lanzada por el sistema. También se indica si se requiere de algún permiso especial y si hay información `EXTRA` que se pasa a través de la intención.

Nombre de la acción /(CONSTANTE)	**Descripción / Permiso** (INFORMACIÓN EXTRA EN INTENT) ☑ ->en lista blanca
Batería	
android.intent.action.BATTERY_LOW (ACTION_BATTERY_LOW)	Batería baja *(Solo sistema).*
android.intent.action.BATTERY_OKAY (ACTION_BATTERY_OKAY)	Batería correcta después de haber estado baja *(Solo sistema).*
android.intent.action.ACTION_POWER_CONNECTED (ACTION_POWER_CONNECTED)	La alimentación se ha conectado *(Solo sistema).*
android.intent.action.ACTION_POWER_DISCONNECTED (ACTION_POWER_DISCONNECTED)	La alimentación se ha desconectado *(Solo sistema).*
android.intent.action.BATTERY_CHANGED (ACTION_BATTERY_CHANGED)	Cambia el estado de la batería (*No Manifest) (Solo sistema).*
Sistema	
android.intent.action.BOOT_COMPLETED (ACTION_BOOT_COMPLETED)	Sistema operativo cargado. Permiso RECEIVE_BOOT_COMPLETED *(Solo sistema).* ☑
android.intent.action.ACTION_SHUTDOWN (ACTION_SHUTDOWN)	El dispositivo va a ser desconectado *(Solo sistema).*
android.intent.action.AIRPLANE_MODE (ACTION_AIRPLANE_MODE_CHANGED)	Modo vuelo activo *(Solo sistema).*
android.intent.action.TIME_TICK (ACTION_TIME_TICK)	Se envía cada minuto (*No Manifest) (Solo sistema).*
android.intent.action.TIME_SET (ACTION_TIME_CHANGED)	La fecha/hora es modificada *(Solo sistema).* ☑

Nombre de la acción /(CONSTANTE)	Descripción / Permiso (INFORMACIÓN EXTRA EN INTENT) ☑->en lista blanca
android.intent.action.CONFIGURATION_CHANGED (ACTION_CONFIGURATION_CHANGED)	Cambia la configuración del dispositivo (orientación, idioma, etc.) *(No Manifest) (Solo sistema).*
Entradas y pantalla	
android.intent.action.SCREEN_OFF (ACTION_SCREEN_OFF)	La pantalla se apaga *(Solo sistema).*
android.intent.action.SCREEN_ON (ACTION_SCREEN_ON)	La pantalla se enciende *(Solo sistema).*
android.intent.action.CAMERA_BUTTON (ACTION_CAMERA_BUTTON)	Se pulsa el botón de la cámara (EXTRA_KEY_EVENT).
android.intent.action.HEADSET_PLUG (ACTION_HEADSET_PLUG)	Se conectan los auriculares (extras: *state, name, microphone*).
android.intent.action.INPUT_METHOD_CHANGED (ACTION_INPUT_METHOD_CHANGED)	Cambia método de entrada.
android.intent.action.USER_PRESENT (ACTION_USER_PRESENT)	El usuario está presente después de que se active el dispositivo *(Solo sistema).*
Memoria y escáner multimedia	
android.intent.action.DEVICE_STORAGE_LOW (ACTION_DEVICE_STORAGE_LOW)	Queda poca memoria *(Solo sistema).*
android.intent.action.DEVICE_STORAGE_OK (ACTION_DEVICE_STORAGE_OK)	Salimos de la condición de poca memoria *(Solo sistema).*
android.intent.action.MEDIA_EJECT (ACTION_MEDIA_EJECT)	El usuario pide extraer almacenamiento externo. ☑
android.intent.action.MEDIA_MOUNTED (ACTION_MEDIA_MOUNTED)	Almacenamiento externo disponible. ☑
android.intent.action.MEDIA_REMOVED (ACTION_MEDIA_REMOVED)	Almacenamiento externo no disponible. ☑
android.intent.action.MEDIA_SCANNER_FINISHED (ACTION_MEDIA_SCANNER_FINISHED)	El escáner de medios termina un directorio (se indica en Intent.mData).
android.intent.action.MEDIA_SCANNER_SCAN_FILE (ACTION_MEDIA_SCANNER_SCAN_FILE)	El escáner de medios encuentra un fichero (se indica en Intent.mData).

Nombre de la acción /(CONSTANTE)	Descripción / Permiso (INFORMACIÓN EXTRA EN INTENT) ✓ ->en lista blanca
android.intent.action.MEDIA_SCANNER_STARTED (ACTION_MEDIA_SCANNER_STARTED)	El escáner de medios comienza un directorio (se indica en Intent.mData).
Aplicaciones	
android.intent.action.MY_PACKAGE_REPLACED (ACTION_MY_PACKAGE_REPLACED)	Una nueva versión de tu aplicación ha sido instalada *(Solo sistema).*
android.intent.action.PACKAGE_ADDED (ACTION_PACKAGE_ADDED)	Una nueva aplicación instalada (EXTRA_UID, EXTRA_REPLACING) *(Solo sistema).*
android.intent.action.PACKAGE_FIRST_LAUNCH (ACTION_PACKAGE_FIRST_LAUNCH)	Primera vez que se lanza una aplicación *(Solo sistema).*
android.intent.action.PACKAGE_REMOVED(ACTION_PACKAGE_REMOVED)	Se desinstala una aplicación *(Solo sistema).*
Comunicaciones y redes	
android.intent.action.PHONE_STATE (ACTION_PHONE_STATE_CHANGED)	Cambia el estado del teléfono. Permiso: READ_PHONE_STATE (EXTRA_STATE, EXTRA_STATE_RINGING). ✓
android.intent.action.NEW_OUTGOING_CALL (ACTION_NEW_OUTGOING_CALL)	Se va a hacer una llamada. Permiso PROCESS_OUTGOING_CALLS (*EXTRA_PHONE_NUMBER*) *(Solo sistema).* ✓
android.provider.Telephony.SMS_RECEIVED	Se recibe un SMS. Permiso: RECEIVE_SMS (Extra: "pdus" ver ejemplo[39]). ✓
android.bluetooth.adapter.action.DISCOVERY_STARTED (ACTION_DISCOVERY_STARTED)	Comienza escáner Bluetooth.
android.bluetooth.adapter.action.STATE_CHANGED (ACTION_STATE_CHANGED)	Bluetooth habilitado/deshabilitado.
android.net.wifi.NETWORK_IDS_CHANGED (NETWORK_IDS_CHANGED_ACTION)	Cambia el identificador de la red Wi-Fi conectada.

[39] stackoverflow.com/questions/33517461/smsmessage-createfrompdu-is-deprecated-in-android-api-level-23

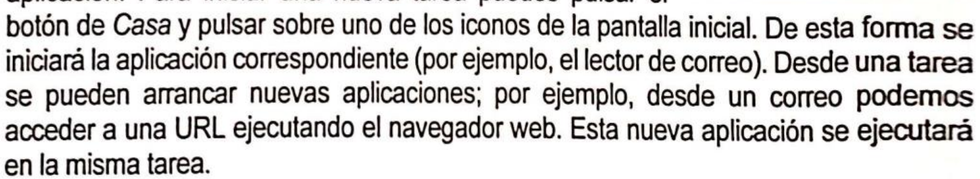


Nombre de la acción /(CONSTANTE)	Descripción / Permiso (INFORMACIÓN EXTRA EN INTENT) ✅->en lista blanca
android.net.wifi.STATE_CHANGE (NETWORK_STATE_CHANGED_ACTION)	Cambia la conectividad Wi-Fi (EXTRA_NETWORK_INFO, EXTRA_BSSID, EXTRA_WIFI_INFO).
android.net.wifi.RSSI_CHANGED (RSSI_CHANGED_ACTION)	Cambia el nivel de señal Wi-Fi (EXTRA_NEW_RSSI).

8.5.3. Arrancar una actividad en una nueva tarea desde un receptor de anuncio

El concepto de tarea no había sido introducido en el curso. Sin embargo, resulta sencillo, y seguro que si eres usuario de Android estás familiarizado con él. La forma más sencilla de entenderlo es que pulses en tu dispositivo móvil el botón cuadrado si tienes la versión 5.0 o superior o el *Casa* durante un segundo, en versiones anteriores. Se mostrará la lista de tareas que hay actualmente en ejecución o que han sido ejecutadas recientemente. Puedes intercambiar de tarea simplemente pulsando sobre una de las previsualizaciones que aparecen en pantalla.

No hay que confundir el concepto de tarea con el de aplicación. Para iniciar una nueva tarea puedes pulsar el botón de *Casa* y pulsar sobre uno de los iconos de la pantalla inicial. De esta forma se iniciará la aplicación correspondiente (por ejemplo, el lector de correo). Desde una tarea se pueden arrancar nuevas aplicaciones; por ejemplo, desde un correo podemos acceder a una URL ejecutando el navegador web. Esta nueva aplicación se ejecutará en la misma tarea.



Otro aspecto a destacar es que cada tarea tiene una pila de actividades independiente. Es decir, si pulsamos el botón de volver en la tarea descrita, pasaremos de nuevo al lector de correo. Pero si cambiamos de tarea y pulsamos el botón de volver, el resultado será muy diferente.



Ejercicio: *Arranque de una actividad al llegar un SMS*

Vamos a modificar el proyecto Asteroides para que se arranque automáticamente la actividad `AcercaDeActivity` al llegar un SMS cualquiera.

1. Abre el proyecto Asteroides.
2. En *AndroidManifest.xml* pide el permiso adecuado y registra el receptor de anuncios:

```
...
<uses-permission android:name ="android.permission.RECEIVE_SMS"/>
...
<application>
   ...
   <receiver android:name="ReceptorSMS" >
      <intent-filter>
         <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
      </intent-filter>
   </receiver>
</application>
...
```

3. Crea una nueva clase con el siguiente código:

```
public class ReceptorSMS extends BroadcastReceiver {
   @Override  public void onReceive(Context context, Intent intent) {
      Intent i = new Intent(context, AcercaDeActivity.class);
      i.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
      context.startActivity(i);
   }
}
```

La forma de arrancar una actividad desde un receptor de anuncios es muy similar a la que hemos estudiado en el capítulo 3. La única diferencia es que ahora hemos necesitado añadir un *flag* a la intención, que indica que la actividad sea creada en una nueva tarea.

4. Ejecuta la aplicación. Envía un SMS al dispositivo y verifica que se abre la actividad *Acerca de ...*

NOTA: Si utilizas el emulador, puedes utilizar la vista EmulatorControl de Android Studio para simular el envío de un SMS.

Cuando lanzamos una nueva actividad, Android nos permite controlar en qué tarea y en qué posición de la pila se situará. No obstante, se recomienda usar siempre el sistema estándar. Es decir, si lanzamos una nueva actividad desde otra actividad, la nueva actividad se sitúa en la misma tarea en la cima de la pila de actividades. Otra cosa es lanzar la actividad desde un receptor de anuncios, dado que cuando llegue el SMS podemos encontrarnos en cualquier tarea. En ese caso, resulta imprescindible activar el *flag* `FLAG_ACTIVITY_NEW_TASK`, así podrá crearse una nueva tarea.



Enlaces de interés:

Lanzar las actividades de la forma estándar suele ser lo más adecuado en la mayoría de los casos. No obstante, si quieres profundizar sobre este tema te recomendamos los siguientes enlaces:

- ***Tasks and Back Stack:*** Documentación oficial de Android.
  http://developer.android.com/guide/components/tasks-and-back-stack.html

- ***Manipulating Android tasks and back stack:*** Presentación didáctica con muchos ejemplos.
  http://es.slideshare.net/RanNachmany/manipulating-android-tasks-and-back-stack

8.5.4. Arrancar un servicio tras cargar el sistema operativo

En muchas ocasiones puede ser interesante que un servicio de nuestra aplicación esté siempre activo, incluso aunque el usuario no haya arrancado nuestra aplicación. Imagina, por ejemplo, un servicio de mensajería instantánea, que ha de estar siempre atento a la llegada de mensajes. Conseguirlo es muy fácil, no tenemos más que crear un receptor de anuncios que se active ante el anuncio `android.intent.action.BOOT_COMPLETED`. Desde este receptor podremos crear el servicio. Para poder registrar el receptor es obligatorio solicitar el permiso `RECEIVE_BOOT_COMPLETED`. Veamos los tres pasos a seguir:

1. Creamos un receptor de anuncios:

```
public class ReceptorArranque extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      context.startForegroundService(new Intent(context, Servicio.class));
   }
}
```

   Desde Android 8 es obligatorio arrancar el servicio en primer plano. Además, al no estar nuestra aplicación en primer plano, no podemos llamar a `startServices()` desde el receptor. Para resolverlo has de usar `startForegroundService()`, con lo que nos comprometemos a pasar el servicio a primer plano en cuanto nos sea posible.

2. Creamos el servicio:

```
public class Servicio extends Service {
   @Override
   public void onCreate() {
      …
      startForeground(NOTIFICACION_ID, notificacion.build());
   }
   @Override
   public int onStartCommand(Intent intencion, int flags, int idArran){…}
   @Override
   public void onDestroy() {…}
   @Override
   public IBinder onBind(Intent intencion) {
      return null;
   }
}
```

3. En *AndroidManifest.xml* pedimos el permiso adecuado y registramos el receptor de anuncios:

```
...
<uses-permission android:name =
                        "android.permission.RECEIVE_BOOT_COMPLETED"/>
<application>
...
   <receiver android:name="ReceptorArranque" >
      <intent-filter >
           <action android:name="android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
   </receiver>
</application>
...
```

NOTA: *En dispositivos de algunas marcas, como Xiaomi, Huawei o HTC, para recibir el anuncio BOOT_COMPLETED es imprescindible ir a opciones de la aplicación y activar el permiso, "Inicio automático". Para abrir automáticamente la configuración consulta:* https://stackoverflow.com/questions/39368251

NOTA: *Solo las aplicaciones instaladas en memoria interna reciben BOOT_COMPLETED. Para forzar que tu aplicación se instale en memoria interna añade el siguiente atributo en la etiqueta <manifest>:*

Android:installLocation="internalOnly"



Práctica: *Arranque automático del servicio de música*

Modifica el proyecto ServicioMusica para que el servicio se active desde el arranque del sistema operativo.



Preguntas de repaso: *Receptores de anuncios*

8.6. Un receptor de anuncios como mecanismo de comunicación

Hasta ahora hemos visto como los receptores de anuncios nos permitían reaccionar ante ciertas circunstancias que ocurrían en el sistema (batería baja, llamada entrante, etc.). En este apartado vamos a ver lo sencillo que resulta crear nuestros propios anuncios *broadcast* y recogerlos desde cualquier componente de nuestra aplicación. Además, estos anuncios también podrán recogerse desde otras aplicaciones.

En un apartado anterior hemos visto cómo asociar anuncios *broadcast* a receptores de anuncios por medio de *AndroidManifest.xml*. En este apartado vamos a realizar la misma tarea utilizando Java. El programador puede escoger uno u otro modo según le convenga.



Ejercicio: *Creación de un nuevo tipo de anuncio* broadcast

En un ejercicio anterior hemos creado un servicio desde una actividad para realizar una operación matemática. Una vez que el servicio ha concluido la operación, queremos que avise a la actividad y le devuelva el valor calculado. En este ejercicio realizaremos este trabajo por medio de un anuncio *broadcast.*

1. Abre el proyecto IntentService creado en el apartado anterior.

2. Añade el siguiente código dentro de la clase `MainActivity`:

```
public class ReceptorOperacion extends BroadcastReceiver {
   public static final String ACTION_RESP =
         "com.example.intentservice.intent.action.RESPUESTA_OPERACION";
   @Override
   public void onReceive(Context context, Intent intent) {
      Double res = intent.getDoubleExtra("resultado", 0.0);
      salida.append(" " + res + "\n");
   }
}
```

Esta nueva clase solo va a utilizarse en esta actividad, por lo que puede definirse dentro de la clase `MainActivity`, en lugar de en un fichero independiente. Se trata de un receptor *broadcast*, que cada vez que llegue un nuevo anuncio leerá un valor enviado en el extra `"resultado"` y lo añadirá al `TextView salida`.

3. Añade las siguientes líneas al método `onCreate()`:

```
IntentFilter filtro = new IntentFilter(ReceptorOperacion.ACTION_RESP);
filtro.addCategory(Intent.CATEGORY_DEFAULT);
registerReceiver(new ReceptorOperacion(), filtro);
```

Con este código hemos asociado el tipo de anuncio *broadcast* a nuestro receptor de anuncios. Como hemos visto en otro apartado, esta tarea también puede realizarse por medio de *AndroidManifest.xml*. El programador puede escoger uno u otro modo según le convenga. Al tratarse de un anuncio para una comunicación interna a nuestra aplicación, parece más conveniente realizarlo así que publicarlo por *AndroidManifest.xml*.

4. Nos queda lanzar el anuncio *broadcast*. Para ello reemplaza la siguiente línea de `IntentServiceOperacion.onHandleIntent()`:

```
MainActivity.salida.append(n*n + "\n");
```

por:

```
Intent i = new Intent();
i.setAction(MainActivity.ReceptorOperacion.ACTION_RESP);
i.addCategory(Intent.CATEGORY_DEFAULT);
i.putExtra("resultado", n*n);
sendBroadcast(i);
```

5. Verifica que la aplicación funciona perfectamente. Pulsa repetidas veces el botón y verifica que esta no se bloquea mientras se calculan las operaciones. Advierte como, aunque se pulse tres veces seguidas, no comienzan las tres operaciones a la vez. Estas serán realizadas de una en una, de manera que irán apareciendo los resultados a intervalos de 5 segundos.

6. Modifica el tiempo de retardo para que este sea de 25 seg. (`sleep(25000)`). Ejecuta de nuevo la aplicación y observa como el sistema no nos muestra ningún error.

8.6.1. Receptores de anuncios desde Android 8

Igual que los servicios, el uso de receptores de anuncios presenta ciertas limitaciones desde Android 8.

Cuando el sistema lanza un anuncio broadcast, decenas de aplicaciones pueden activarse. Si el receptor está definido en el manifiesto, se puede incluso lanzar la aplicación si esta no está en memoria. Esto puede bloquear el programa actual, empeorando la experiencia de usuario.

Desde Android 8 solo se podrán definir receptores en el manifiesto, si son anuncios creados por nosotros o están en la lista blanca. Los receptores de anuncios del sistema, definidos en el manifiesto, dejaran de recibir notificaciones a no ser que estén en la lista blanca:

https://developer.android.com/guide/components/broadcast-exceptions

Google prefiere que registremos los anuncios dinámicamente por código y quitarlos al pasar a segundo plano.

8.6.2. Anuncios *broadcast* permanentes

Android permite enviar dos tipos de anuncios *broadcast*, normales y permanentes. Un *broadcast* permanente llegará a los receptores de anuncios que actualmente estén escuchando, pero también a los que se instancien en un futuro. Por ejemplo, el sistema emite el anuncio *broadcast* `ACTION_BATTERY_CHANGED` de forma permanente. De esta forma, cuando se llama a `registerReceiver()` se obtiene la intención de la última emisión de este anuncio. Por lo tanto, puede usarse para encontrar el estado de la batería sin necesidad de esperar a un futuro cambio en su estado. Este tipo de anuncios también se conocen como persistentes o pegajosos (del término en inglés *sticky*).

Para enviar un *broadcast* permanente utiliza el método `sendStickyBroadcast()` en lugar de `sendBroadcast()`:

```
Intent i = new Intent("com.example.intent.action.ACCION");
sendStickyBroadcast(i);
```

Enviar un anuncio permanente requiere de la solicitud del siguiente permiso:

```
<uses-permission android:name="android.permission.BROADCAST_STICKY"/>
```

Se exige este permiso dado que las aplicaciones mal intencionadas pueden ralentizar el dispositivo o volverlo inestable al demandar demasiada memoria.



Preguntas de repaso: *Receptores anuncios para la comunicación*

8.7. Un servicio como mecanismo de comunicación entre aplicaciones

Como hemos comentado, un servicio tiene una doble funcionalidad: además de permitir la ejecución de código en segundo plano, vamos a poder utilizarlo como un mecanismo de comunicación entre aplicaciones[40]. Cuando una aplicación quiere compartir algún tipo de información con otra aplicación, se presenta un problema. Las aplicaciones en Android se ejecutan en procesos separados y, por tanto, tienen espacios de memoria distintos. Esto nos impide, por ejemplo, que ambas aplicaciones compartan un mismo objeto.

Como respuesta a este problema, Android nos propone un mecanismo de comunicación entre procesos que se basa en un lenguaje de especificación de interfaces, AIDL (*Android Interface Definition Language*). Las interfaces AIDL se publican por medio de servicios. Gracias al lenguaje de especificación de interfaces AIDL, un proceso en Android puede llamar a un método de un objeto situado en un proceso diferente del suyo. Se trata de un mecanismo de comunicación entre procesos similar a COM o Corba, aunque algo más ligero.

Si queremos comunicar dos aplicaciones a través de este mecanismo, seguiremos los siguientes pasos:

1. Escribiremos un fichero AIDL: En él se define la interfaz, es decir, los métodos y los parámetros que luego podremos utilizar.
2. Implementaremos los métodos de la interfaz: Para ello habrá que crear una clase en Java que implemente estos métodos.
3. Publicar la interfaz a los clientes: Para ello se extenderá la clase `Service` y sobrescribiremos el método `onBind(Intent)` de forma que devuelva una instancia de la clase que implementa la interfaz.



Vídeo[tutorial]: *Un servicio como mecanismo de comunicación entre aplicaciones*

[40] http://developer.android.com/guide/topics/fundamentals.html#rpc

Veamos estos tres pasos más detenidamente por medio de un ejemplo. Para ello crea la siguiente aplicación:

```
Template / Phone and Tablet / Empty Activity
Application Name: Servicio Remoto
Package Name: org.example.servicioremoto
Minimum SDK: API 16 Android 4.1 (Jel Bean)
```

Reemplaza el código del *layout* `activity_main.xml` por el mismo utilizado en `ServicioMusica`. Reemplaza los textos de los botones *Arrancar servicio* por *Conectar servicio* y *Detener servicio* por *Desconectar servicio.* Crea dos botones más. Uno con texto "Reproducir" e *id* "@+id/boton_reproducir" y otro con texto "Avanzar" e *id* "@+id/boton_avanzar".

Copia el fichero `res/raw/audio.mp3` en la nueva aplicación.

8.7.1. Crear la interfaz en AIDL

El lenguaje de especificación de interfaces AIDL tiene una sintaxis similar a Java, aunque como su nombre indica, permite únicamente identificar la interfaz de un objeto, no su implementación.

Una interfaz está formada por una secuencia de métodos, cada uno con una serie de parámetros y un valor devuelto. Tanto los parámetros como el valor devuelto han de tener un tipo. Los tipos permitidos se indican a continuación:

- Tipos primitivos: `int`, `short`, `byte`, `char`, `float`, `double`, `long`, `boolean`.
- Una de las siguientes clases: `String`, `CharSequence`, `List`, `Map`.
- Una interfaz escrita en AIDL.
- Una subclase de `Parcelable`.

Para seguir con el ejemplo, crea un nuevo fichero que se llame `IServicioMusica.aidl` dentro de `org.example.servicioremoto/` con el siguiente código:

```
package org.example.servicioremoto;

interface IServicioMusica {
    String reproduce(in String mensaje);
    void setPosicion(int ms);
    int getPosicion();
}
```

Como puedes observar, la sintaxis es similar a Java, aunque existen diferencias. La más destacable consiste en que los parámetros de los métodos cuyos tipos no sean primitivos han de indicar la etiqueta `<in>`, `<out>` o `<inout>`, según sean parámetros de entrada, salida o las dos cosas a la vez. Para los tipos primitivos solo se permite que actúen como entrada; por lo tanto, se procesan de forma predeterminada como `<in>`.

NOTA: *Igual como ocurre con las clases en Java, las interfaces en AIDL han de escribirse en un fichero con el mismo nombre que la interfaz.*

8.7.2. Implementar la interfaz

Una vez escrita esta interfaz y almacenada en el fichero `.aidl`, se generará de forma automática el fichero `app/build/generated/aidl_source_output_dir/debug/out/org.example.servicioremoto/ IServicioMusica.java`. Para ver este fichero, selecciona en el explorador del proyecto, en el desplegable de arriba, el valor *Project*, en lugar de *Android*.

Este fichero define la interfaz Java `IServicioMusica` como descendiente de `android.os.IInterface`. En `IServicioMusica` se define internamente la clase abstracta `Stub` que implementa la interfaz escrita en AIDL. Es decir, la clase `Stub` contiene tantos métodos abstractos como métodos declaramos en la interfaz AIDL. La característica especial de estos métodos es que podrán ser invocados desde un proceso remoto. Es decir, podrán ser usados para el intercambio de información entre procesos.

Ahora tenemos que darle funcionalidad a la interfaz. Para ello hay que crear una clase que extienda `IServicioMusica.Stub` y que implemente todos los métodos abstractos de esta clase, o lo que es lo mismo, los métodos declarados en la interfaz AIDL. Un ejemplo de cómo podríamos implementar estos métodos se muestra a continuación. Más tarde se indica dónde hay que introducir este código:

```
private final IServicioMusica.Stub binder = new IServicioMusica.Stub() {

   public String reproduce(String mensaje) {
      reproductor.start();
      return mensaje;
   }

   public void setPosicion(int ms) {
      reproductor.seekTo(ms);
   }

   public int getPosicion() {
      return reproductor.getCurrentPosition();
   }
};
```

La variable `reproductor` será declarada posteriormente de tipo `MediaPlayer`. Como puedes ver, en el método `reproduce`, tanto el parámetro de entrada como el valor devuelto no tienen ninguna utilidad. Se han introducido para ilustrar el paso de una variable no primitiva.

Cuando implementes los métodos de una interfaz AIDL has de tener en cuenta lo siguiente:

- Si generas una excepción desde uno de estos métodos, esta no pasará a la aplicación que hizo la llamada.
- Las llamadas son síncronas. Por lo tanto, has de tener cuidado de que cuando se haga una llamada que tarde cierto tiempo en responder, nunca se haga desde el hilo principal de la aplicación. Si el hilo principal queda bloqueado demasiado tiempo, aparecerá el incómodo cuadro de diálogo "La aplicación no responde". En estos casos, crea un hilo secundario, desde donde se haga la llamada.

- Solo es posible declarar métodos; no puedes declarar campos estáticos en una interfaz AIDL.

8.7.3. Publicar la interfaz en un servicio

Otras aplicaciones han de tener visible nuestra interfaz para poder comunicarse con nosotros. Esto se consigue publicando un servicio que contenga nuestra interfaz.

Para ello has de crear una clase que herede de `Service` y que rescriba el método `onBind()`. Este método se utilizará para devolver un objeto que implementa nuestra interfaz. Veamos cómo se escribiría este servicio:

```
public class ServicioRemoto extends Service {
    MediaPlayer reproductor;
    @Override public void onCreate() {
        super.onCreate();
        reproductor = MediaPlayer.create(ServicioRemoto.this, R.raw.audio);
    }
    private final IServicioMusica.Stub binder = new IServicioMusica.Stub() {
        // Copia aquí el código anterior
    };
    @Override public IBinder onBind(Intent intent) {
        return this.binder;
    }
    @Override public boolean onUnbind(Intent intent) {
        reproductor.stop();
        return true;
    }
}
```

Crea una nueva clase en el proyecto e introduce este código. Para que el servicio sea visible a otras aplicaciones hay que publicarlo declarándolo en *AndroidManifest.xml*. Para ello, copia el siguiente código dentro de la etiqueta `<application>`:

```
<service android:name=".ServicioRemoto" android:process=":remote">
    <intent-filter>
        <action android:name="org.example.servicioremoto.IServicioMusica"/>
    </intent-filter>
</service>
```

El atributo `name` ha de coincidir con la clase que implementa el servicio. El atributo `process` permite que el servicio se ejecute en un proceso propio, diferente del resto de los componentes de la aplicación. A continuación indicaremos dentro de la etiqueta `<intent-filter>` una etiqueta `<action>` por cada interfaz que queremos publicar. Un servicio podría publicar más de una interfaz, para lo que habría que escribir un fichero AIDL por cada interfaz.

8.7.4. Llamar a una interfaz remota

Para llamar a la interfaz creada anteriormente, sigue los siguientes pasos:

1. Declara una variable de tipo `IServicioMusica`. Esta variable se utilizará para hacer llamadas al objeto remoto.
2. Implementa la interfaz `ServiceConnection`. Los escuchadores de esta interfaz nos permitirán controlar cuando se produce una conexión (`onServiceConnected()`) y cuando se produce una desconexión del servicio (`onServiceDisconnected()`).
3. Para conectarte al servicio llama al método `bindService()`, pasándole un objeto de la clase `ServiceConnection` que acabas de implementar en el paso 2.
4. Si se puede realizar la conexión, se llamará al método `ServiceConnection.onServiceConnected()`, que recibirá en uno de sus parámetros una instancia de `IBinder`. Utiliza el método `IServicioMusica.Stub.asInterface()`, pasándole como parámetro esta instancia de `IBinder` para inicializar la variable declarada en el primer punto.
5. Ya puedes llamar a los métodos del objeto remoto declarados en el punto 1. En nuestro caso, `reproducir()`, `setPosicion()` y `getPosicion()`.
6. Puedes desconectarte del servicio utilizando el método `unbindService()`.

Realizaremos estos pasos en la actividad principal de la aplicación:

```
public class MainActivity extends Activity {
   private IServicioMusica servicio;                              (1)
   private ServiceConnection conexion = new ServiceConnection() {  (2)
      public void onServiceConnected(ComponentName className,      (4)
                                     IBinder iservicio) {
         servicio = IServicioMusica.Stub.asInterface(iservicio);
         Toast.makeText(MainActivity.this,
                  "Conectado a Servicio", Toast.LENGTH_SHORT).show();
      }
      public void onServiceDisconnected(ComponentName className) {
         servicio = null;
         Toast.makeText(MainActivity.this,
                  "Se ha perdido la conexión con el Servicio",
                  Toast.LENGTH_SHORT).show();
      }
   };
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout. activity_main);
      Button botonConectar = findViewById(R.id.boton_arrancar);

      botonConectar.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
```

```
            MainActivity.this.bindService(                          (3)
               new Intent(MainActivity.this, ervicioRemoto.class),
                     conexion, Context.BIND_AUTO_CREATE);
         }
      });
      Button botonReproducir = findViewById(R.id.boton_reproducir);
      botonReproducir.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
            try { servicio.reproduce("titulo");                     (5)
            } catch (Exception e) {
               Toast.makeText(MainActivity.this, e.toString(),
                                       Toast.LENGTH_SHORT).show();
            }
         }
      });
      Button botonAvanzar = findViewById(R.id.boton_avanzar);
      botonAvanzar.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
            try { servicio.setPosicion(servicio.getPosicion()+1000);  (5)
            } catch (Exception e) {
               Toast.makeText(MainActivity.this, e.toString(),
                                       Toast.LENGTH_SHORT).show();
            }
         }
      });
      Button botonDetener = findViewById(R.id.boton_detener);
      botonDetener.setOnClickListener(new OnClickListener() {
         public void onClick(View v) {
            try{ MainActivity.this.unbindService(conexion);         (6)
            } catch (Exception e) {
               Toast.makeText(MainActivity.this, e.toString(),
                                       Toast.LENGTH_SHORT).show();
            }
            servicio = null;
         }
      });
   }
}
```

La actividad está formada por 4 botones, para realizar las acciones de conectarse al servicio, desconectarse, reproducir música y avanzar 1 segundo (1.000 ms). Si los botones no se activan en el orden adecuado podemos provocar excepciones. Estas son capturadas y visualizadas mediante un `Toast`, por lo que la aplicación continuará funcionando, aunque se produzcan. Prueba a llamar a un método antes de establecer la conexión o trata de desconectarte dos veces para observar las excepciones generadas.



Práctica: *Servicio de detección de caídas*

En esta práctica se va a repasar el uso de los siguientes componentes: **layouts**, **actividades**, **servicios**, **notificaciones**, **sensores**, **intenciones** y **receptores de anuncios**.

1. Crea un proyecto de tipo "Empty Activity" llamado "ServicioCaidas".
2. En la actividad principal añade un botón "Arrancar servicio detección de aceleración" y otro "Detener servicio".
3. Crea un servicio que se arranque y se detenga al pulsar los botones anteriores. En el servicio habrá un Toast que indique si se ha arrancado o detenido.
4. Haz que este servicio escuche el sensor de aceleración. Cuando se lea un valor se mostrará usando Log.d("CAIDAS", "...") para cada uno de los tres ejes;
5. Haz que solo se muestre el Log cuando la aceleración en alguno de los ejes supere un determinado umbral. Ajústalo para que solo aparezca el Log al desplazar muy fuerte el teléfono.
6. Haz que el servicio pase a primer plano.
7. Cuando se pulse sobre la notificación asociada del servicio se abrirá la actividad principal.
8. Cuando se supere el umbral de aceleración haz que el teléfono haga una llamada a un número. Para ello puedes usar una intención ACTION_DIAL o ACTION_CALL.

NOTA: *Cuando lanzas una actividad desde fuera de una actividad es necesario poner a la intención el siguiente flag.*

```
intent.setFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
```

9. Cuando el teléfono es desconectado de corriente haz que muestre un Toast indicándolo.
10. Además de mostrar el Toast haz que se detenga el servicio de caídas. Para tener acceso al servicio crea el receptor de anuncios por código desde el servicio.

NOTA: *Se ha escogido el anuncio de desconexión de la corriente para poder probarlo de forma más sencilla. En la aplicación real sería más interesante detener el servicio cuando la batería esté baja.*

11. Cuando la aceleración supere un umbral, además del LogCat y la llamada, haz que se envíe un anuncio broadcast con el valor de la aceleración. El anuncio ha de ser recogido por la actividad, que mostrará esta información en un TextView.

CAPÍTULO 9.
Almacenamiento de datos

Las aplicaciones descritas hasta este capítulo representan la información a procesar en forma de variables. El problema con las variables es que dejan de existir en el momento en que la aplicación es destruida. En muchas ocasiones vamos a necesitar almacenar información de manera permanente. Las alternativas más habituales para conservar esta información son los ficheros, las bases de datos o servicios a través de la red. Estas técnicas no solo permiten mantener a buen recaudo los datos de la aplicación, si no que también permiten compartir estos datos con otras aplicaciones y usuarios. De forma adicional, el sistema Android pone a nuestra disposición dos nuevos mecanismos para almacenar datos, las preferencias y ContentProvider.

A lo largo de este capítulo estudiaremos cómo utilizar estas técnicas en Android. Comenzaremos describiendo el uso de las preferencias como un mecanismo sencillo para guardar de forma permanente algunas variables. Seguiremos describiendo las características del sistema de ficheros que incorpora Android. Se puede acceder a los ficheros a través de las clases estándar incluidas en Java. De forma adicional se incluyen nuevas clases para cubrir las peculiaridades de Android.

Como tercera alternativa se estudiará el uso de XML para almacenar la información de manera estructurada. Similar a XML tenemos JSON, que presenta la ventaja de usar un formato más compacto. Como quinta alternativa al almacenamiento de datos se estudiarán las bases de datos. Android incorpora la librería SQLite, que nos permitirá crear y manipular nuestras propias bases de datos de forma muy sencilla. Para finalizar, se describirá la clase `ContentProvider`, que consiste en un mecanismo introducido en Android para poder compartir datos entre aplicaciones.

En el capítulo siguiente se describe otra alternativa, el uso de Internet como recurso para almacenar y compartir información. Concretamente se describirá el uso de *sockets* TCP, HTML y los servicios web.

Todas estas alternativas se ilustrarán a través del mismo ejemplo. Trataremos de almacenar la lista con las mejores puntuaciones obtenidas en Asteroides, tal como se ha descrito en el capítulo 3.



Objetivos:

- Repasar las alternativas para el almacenamiento de datos en Android.
- Describir el uso de ficheros.
- Utilizar dos herramientas para manipular ficheros XML (SAX y DOM) y dos herramientas para manipular ficheros JSON (GSON y org.json).
- Mostrar como desde Android podemos utilizar SQLite para trabajar con bases de datos relacionales.
- Describir qué es un ContentProvider y cómo podemos utilizar algunos ContentProvider disponibles en Android.
- Aprender a crear nuestros propios ContentProvider.

9.1. Alternativas para guardar datos permanentemente en Android

Existen muchas alternativas para almacenar información de forma permanente en un sistema informático. A continuación, mostramos una lista de las más habituales utilizadas en Android:

- **Preferencias:** Es un mecanismo liviano que permite almacenar y recuperar datos primitivos en forma de pares clave/valor. Este mecanismo se suele utilizar para almacenar los parámetros de configuración de una aplicación.
- **Ficheros:** Puedes almacenar los ficheros en la memoria interna del dispositivo o en un medio de almacenamiento externo, como una tarjeta SD. También puedes utilizar ficheros añadidos a tu aplicación, como recursos.
- **XML:** Se trata de un estándar fundamental para la representación de datos, en Internet y en muchos otros entornos (como en el Android SDK). En Android disponemos de las librerías SAX y DOM para manipular datos en XML.
- **JSON:** Es una alternativa a XML para almacenar información estructurada. Usa una representación simple y compacta, lo que la hace especialmente interesante para transacciones por Internet. En este capítulo se describen dos herramientas: GSON y org.json.
- **Base de datos:** Las API de Android contienen soporte para SQLite. Tu aplicación puede crear y usar bases de datos SQLite de forma muy sencilla y con toda la potencia que nos da el lenguaje SQL.
- **Proveedores de contenido:** Un proveedor de contenido es un componente de una aplicación que expone el acceso de lectura/escritura de sus datos a otras aplicaciones. Está sujeto a las restricciones de seguridad que quieras imponer. Los proveedores de contenido implementan una sintaxis estándar para acceder a sus datos mediante URI (Uniform Resource Identifiers) y un mecanismo de acceso para devolver los datos similar a SQL. Android

provee algunos proveedores de contenido para tipos de datos estándar, tales como contactos personales, ficheros multimedia, etc.

- **Internet:** No te olvides de que también puedes usar la nube para almacenar y recuperar datos. Se estudia en el siguiente capítulo.



Vídeo[tutorial]: *Almacenamiento de datos en Android*

9.2. Añadiendo puntuaciones en Asteroides

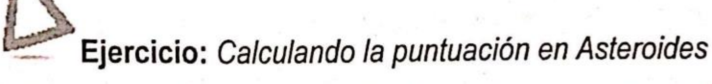
A modo de ejemplo se va a implementar la posibilidad de guardar las mejores puntuaciones obtenidas en Asteroides. Se utilizarán mecanismos alternativos que se desarrollarán a lo largo de este capítulo y el siguiente:

- *Array* (implementado en el capítulo 3)
- Preferencias
- Ficheros en memoria interna, externa y en recursos
- XML con SAX y DOM
- JSON con GSon y org.json
- Base de datos SQLite y con varias tablas relacionales
- ContentProvider
- Internet a través de *sockets*
- Servicios web

Para facilitar la sustitución del método de almacenamiento, en el capítulo 3 se ha creado la siguiente interfaz en la aplicación Asteroides:

```
public interface AlmacenPuntuaciones {
   public void guardarPuntuacion(int puntos, String nombre, long fecha);
   public List<String> listaPuntuaciones(int cantidad);
}
```

También se ha declarado la variable `almacen` de tipo `AlmacenPuntuaciones` y se ha creado la actividad `Puntuaciones`, que visualiza un `ListView` con las puntuaciones. Dado que en el capítulo 3 todavía no teníamos la opción de jugar, no se podían añadir nuevas puntuaciones a `almacen`. En el siguiente ejercicio trataremos de calcular una puntuación en el juego y almacenarla en `almacen`.



Ejercicio: *Calculando la puntuación en Asteroides*

1. Crea una variable global en la clase `VistaJuego` que se llame `puntuacion` e inicialízala a cero:

```
private int puntuacion = 0;
```

2. Cada vez que se destruya un asteroide hay que incrementar esta variable. Añade dentro de `destruyeAsteroide()` la siguiente línea:

```
puntuacion += 1000;
```

3. Cuando desde la actividad inicial `Asteroides` se llame a la actividad `Juego`, nos interesa que esta nos devuelva la puntuación obtenida. Recuerda que en el capítulo 3 hemos estudiado la comunicación entre actividades. Para pasar la información entre las actividades, añade el siguiente código en `Asteroides` en sustitución del método `lanzarJuego()` anterior:

```
static final int ACTIV_JUEGO = 0;

public void lanzarJuego(View view) {
   Intent i = new Intent(this, Juego.class);
   startActivityForResult(i, ACTIV_JUEGO);
}

@Override protected void onActivityResult (int requestCode,
                                   int resultCode, Intent data){
   super.onActivityResult(requestCode, resultCode, data);
   if (requestCode== ACTIV_JUEGO && resultCode==RESULT_OK && data!=null) {
      int puntuacion = data.getExtras().getInt("puntuacion");
      String nombre = "Yo";
      // Mejor leer nombre desde un AlertDialog.Builder o preferencias
      almacen.guardarPuntuacion(puntuacion, nombre,
                                   System.currentTimeMillis());
      lanzarPuntuaciones(null);
   }
}
```

4. Para realizar la respuesta de la actividad será más sencillo hacerlo desde `VistaJuego` que desde `Juego`. El problema es que esta clase es una vista, no una actividad. Para solucionar el problema puedes usar el siguiente truco. Introduce en `VistaJuego` el siguiente código:

```
private Activity padre;

public void setPadre(Activity padre) {
   this.padre = padre;
}
```

5. Cuando se detecte una condición de victoria o derrota, es un buen momento para almacenar la puntuación y salir de la actividad. Para ello crea el siguiente método dentro de `VistaJuego`:

```
private void salir() {
   Bundle bundle = new Bundle();
   bundle.putInt("puntuacion", puntuacion);
   Intent intent = new Intent();
   intent.putExtras(bundle);
   padre.setResult(Activity.RESULT_OK, intent);
   padre.finish();
}
```

6. Al final del método `destruyeAsteroide()` introduce:

```
if (asteroides.isEmpty()) {
   salir();
}
```

7. Al final del método `actualizaFisica()` introduce:

```
for (Grafico asteroide : asteroides) {
   if (asteroide.verificaColision(nave)) {
      salir();
   }
}
```

8. En el método `onCreate` de `Juego` introduce:

```
vistaJuego.setPadre(this);
```

9.2.1. Fragmentando los asteroides

Siguiendo con el juego Asteroides, queremos que cuando el misil alcance un asteroide, este se divida en varios fragmentos. Para conseguirlo puedes seguir las instrucciones del siguiente ejercicio:



Ejercicio: *Fragmentando los asteroides*

1. Convierte la variable local `drawableAsteroide` declarada en el constructor de la clase `VistaJuego` en una variable global, que será un *array* de tres elementos:

```
private Drawable drawableAsteroide[]= new Drawable[3];
```

2. En el constructor, cuando se quiera trabajar con *bitmaps* inicializaremos esta variable de la siguiente forma:

```
drawableAsteroide[0] =
            AppCompatResources.getDrawable(context, R.drawable.asteroide1);
drawableAsteroide[1] =
            AppCompatResources.getDrawable(context, R.drawable.asteroide2);
drawableAsteroide[2] =
            AppCompatResources.getDrawable(context, R.drawable.asteroide3);
```

3. Y en caso de querer trabajar con gráficos vectoriales:

```
for (int i=0; i<3; i++) {
   ShapeDrawable dAsteroide = new ShapeDrawable(new PathShape(
                                          pathAsteroide, 1, 1));
   dAsteroide.getPaint().setColor(Color.WHITE);
   dAsteroide.getPaint().setStyle(Style.STROKE);
   dAsteroide.setIntrinsicWidth(50 - i * 14);
   dAsteroide.setIntrinsicHeight(50 - i * 14);
   drawableAsteroide[i] = dAsteroide;
}
```




4. Añade al principio del método `destruyeAsteroide(int i)` el código:

```
if(asteroides.get(i).getDrawable()!=drawableAsteroide[2]){
   int tamaño;
   if(asteroides.get(i).getDrawable()==drawableAsteroide[1]){
      tamaño =2;
   } else {
      tamaño =1;
   }
   for(int n=0;n<numFragmentos;n++){
      Grafico asteroide = new Grafico(this,drawableAsteroide[tamaño]);
      asteroide.setCenX(asteroides.get(i).getCenX());
      asteroide.setCenY(asteroides.get(i).getCenY());
      asteroide.setIncX(Math.random()*4-2);
      asteroide.setIncY(Math.random()*4-2);
      asteroide.setAngulo((int)(Math.random()*360));
      asteroide.setRotacion((int)(Math.random()*8-4));
      asteroides.add(asteroide);
   }
}
```

Pasemos a explicar el código. Los asteroides tienen tres tamaños, de mayor a menor 0, 1 y 2. Cuando un asteroide de tamaño 0 es destruido, este desaparece pero aparecen varios (`numFragmentos`) de tamaño 1. Cuando uno de tamaño 1 es destruido pasa lo mismo, pero ahora los fragmentos son de tamaño 2. Los asteroides de tamaño 2 son destruidos definitivamente.

Comenzamos verificando si el asteroide destruido es de tamaño 2. En tal caso, no hemos de hacer nada, el resto del método destruye el asteroide. Si no lo es, calculamos en la variable `tamaño` el nuevo tamaño de los fragmentos. Luego hacemos un bucle para añadir los nuevos asteroides.

5. Corrige algún error adicional ocasionado por este cambio.
6. Prueba los cambios propuestos.



Práctica: *Mejorando preferencias en Asteroides (II)*

Modifica el programa para que el número de fragmentos generados corresponda al valor introducido en las preferencias.

9.3. Preferencias

Las preferencias (clase `SharedPreferences`) pueden usarse como un mecanismo para que los usuarios modifiquen algunos parámetros de configuración de la aplicación. Este uso se ha estudiado en el capítulo 3, donde se describe cómo podemos crear una actividad descendiente de `PreferenceFragment` para que el usuario consulte y modifique estas preferencias.

Las preferencias también pueden utilizarse como un mecanismo liviano para almacenar ciertos datos que tu aplicación quiera conservar de forma permanente. Es un mecanismo sencillo que te permite almacenar una serie de variables con su nombre y su valor. Puedes almacenar variables de tipo `boolean`, `int`, `long`, `float` y `String`. En este apartado describimos su utilización.

Las preferencias son almacenadas en ficheros XML dentro de la carpeta `shared_prefs` en los datos de la aplicación. Recuerda que en el capítulo 3 hemos visto que las preferencias de usuario siempre se almacenan en el fichero *paquete*`_preferences`, donde el paquete ha de ser reemplazado por el paquete de la aplicación (en Asteroides, el fichero es `org.example.asteroides_preferences`). Cuando utilices las preferencias para almacenar otros valores, podrás utilizar otros ficheros. Tienes dos alternativas según utilices uno de los siguientes métodos:

- `getSharedPreferences()`: Te permite indicar de forma explícita el nombre de un fichero de preferencias en un parámetro. Puedes utilizarlo cuando necesites varios ficheros de preferencias o acceder al mismo fichero desde varias actividades.
- `getPreferences()`: No tienes que indicar ningún nombre de fichero. Puedes utilizarlo cuando solo necesites un fichero de preferencias en la actividad.

Estos dos métodos necesitan como parámetro el tipo de acceso que queramos dar al fichero de preferencias. Los valores posibles son `MODE_PRIVATE`, `MODE_WORLD_READABLE` o `MODE_WORLD_WRITEABLE` según queramos tener acceso exclusivo a nuestra aplicación, permitir la lectura o permitir la lectura y la escritura a otras aplicaciones.

Una llamada a uno de estos dos métodos te devolverá un objeto de la clase `SharedPreferences`.

Para escribir las preferencias puedes utilizar el siguiente código:

```
SharedPreferences preferencias= getPreferences(MODE_PRIVATE);
SharedPreferences.Editor editor = preferencias.edit();
editor.putString("nombre", "Juan");
editor.putInt("edad", 35);
editor.apply();
```

Para leer las preferencias puedes utilizar el siguiente código:

```
SharedPreferences preferencias= getPreferences(MODE_PRIVATE);
 String nombre = preferencias.getString("nombre",
                                     "valor por defecto");
int edad = preferencias.getInt("edad", -1);
```

El ejemplo anterior puede ser modificado reemplazando `getPreferences()` por `getSharedPreferences()`. En este caso, tendrás que indicar el fichero donde se almacenarán las preferencias.



Vídeo[tutorial]: *Almacenar información usando Preferencias*



Ejercicio: *Almacenando la última puntuación en un fichero de preferencias*

Veamos un ejemplo de cómo podemos crear un fichero de preferencias para almacenar la última puntuación obtenida en Asteroides.

1. Abre el proyecto Asteroides.
2. Crea una nueva clase `AlmacenPuntuacionesPreferencias`.
3. Reemplaza el código por el siguiente:

```
public class AlmacenPuntuacionesPreferencias implements
AlmacenPuntuaciones {
   private static String PREFERENCIAS = "puntuaciones";
   private Context context;
   public AlmacenPuntuacionesPreferencias(Context context) {
      this.context = context;
   }

   public void guardarPuntuacion(int puntos, String nombre,
                                              long fecha) {
      SharedPreferences preferencias =context.getSharedPreferences(
            PREFERENCIAS, Context.MODE_PRIVATE);
      SharedPreferences.Editor editor = preferencias.edit();
      editor.putString("puntuacion", puntos + " " + nombre);
      editor.apply();
   }

   public List<String> listaPuntuaciones(int cantidad) {
      List<String> result = new ArrayList<String>();
      SharedPreferences preferencias =context.getSharedPreferences(
                              PREFERENCIAS, Context.MODE_PRIVATE);
      String s = preferencias.getString("puntuacion", "");
      if (!s.isEmpty()) {
         result.add(s);
      }
      return result;
   }
}
```

4. Abre el fichero *MainActivity.java* y modifica el método `onCreate()` para que la variable `almacen` se inicialice de la siguiente manera:

```
almacen = new AlmacenPuntuacionesPreferencias(this);
```

5. Ejecuta el proyecto y verifica que la última puntuación se guarda correctamente.

6. Selecciona la pestaña *Device File Explorer* en la esquina inferior derecha. Verifica que se ha creado el fichero */data/data/org.example.asteroides/shared_prefs/puntuaciones.xml*.

7. Haz doble clic sobre el fichero y observa su contenido.



Práctica: *Almacenando las últimas 10 puntuación en un fichero de preferencias*

En el ejercicio anterior solo hemos guardado la última puntuación, lo cual no coincide con la idea planteada en un principio: nos interesaba guardar una lista con las últimas puntuaciones. En esta práctica has de tratar de solucionar este inconveniente. *NOTA: Se trata de una práctica básicamente de programación en Java. Si no estás interesado puedes consultar directamente la solución.*

1. Las preferencias solo están preparadas para almacenar variables de tipos simple, por lo que no permiten almacenar una lista. Para solucionar este inconveniente, te recomendamos que crees 10 preferencias que se llamen `puntuacion0`, `puntuacion1`, ... , `puntuacion9`.

2. Cuando se llame a `guardarPuntuacion()` almacena la nueva puntuación en `puntuacion0`. Pero antes ten la precaución de copiar el valor de `puntuacion8` en `puntuacion9`; `puntuacion7` en `puntuacion8`, y así hasta la primera. Esta operación puede realizarse por medio de un bucle con un índice entero, `n`, de forma que el nombre de la preferencia a mover puedes expresarlo como `"puntuacion"+n`.

3. Utiliza el mismo truco para implementar el método `listaPuntuaciones()`.



Solución: *Almacenando las últimas 10 puntuación en un fichero de preferencias*

1. Reemplaza en `guardarPuntuacion()`:

```
editor.putString("puntuacion", puntos + " " + nombre);
```

por:

```
for (int n = 9; n >= 1; n--) {
   editor.putString("puntuacion" + n,
               preferencias.getString("puntuacion" + (n - 1), ""));
}
editor.putString("puntuacion0", puntos + " " + nombre);
```

2. Reemplaza en `listaPuntuaciones()`:

```
String s = preferencias.getString("puntuacion", "");
if (!s.isEmpty()) {
   result.add(s);
}
```

por:

```
for (int n = 0; n <= 9; n++) {
   String s = preferencias.getString("puntuacion" + n, "");
   if (!s.isEmpty()) {
      result.add(s);
   }
}
```

9.4. Accediendo a ficheros

Existen tres tipos de ficheros donde podemos almacenar información en Android: ficheros almacenados en la memoria interna del teléfono, ficheros almacenados en la memoria externa (normalmente una tarjeta SD) y ficheros almacenados en los recursos. Estos últimos son de solo lectura, por lo que no son útiles para almacenar información desde la aplicación. Cuando programes en Android debes tener en cuenta que un dispositivo móvil tiene una capacidad de almacenamiento limitada.



Vídeo[tutorial]: *Gestión de ficheros en Android*

9.4.1. Sistema interno de ficheros

Android permite almacenar ficheros en la memoria interna del teléfono. Por defecto, los ficheros almacenados solo son accesibles para la aplicación que los creó, no pueden ser leídos por otras aplicaciones, ni siquiera por el usuario del teléfono. Cada aplicación dispone de una carpeta especial para almacenar ficheros (*/data/data/nombre_del_paquete/files*). La ventaja de utilizar esta carpeta es que cuando se desinstala la aplicación los ficheros que has creado se eliminarán. Cuando trabajes con ficheros en Android, ten siempre en cuenta que la memoria disponible de los teléfonos móviles es limitada.

Recuerda que el sistema de ficheros se sustenta en la capa Linux, por lo que Android hereda su estructura. Cuando se instala una nueva aplicación, Android crea un nuevo usuario Linux asociado a la aplicación y es este usuario el que podrá o no acceder a los ficheros.

Puedes utilizar cualquier rutina del paquete `java.io` para trabajar con ficheros. Adicionalmente se han creado métodos adicionales asociados a la clase `Context` para facilitarte el trabajo con ficheros almacenados en la memoria interna. En particular, los métodos `openFileInput()` y `openFileOutput()` te permiten abrir un fichero para lectura o escritura respectivamente. Si utilizas estos métodos, el nombre del archivo no puede contener subdirectorios. De hecho, el fichero siempre se almacena en la carpeta reservada para tu aplicación (*/data/data/nombre_del_paquete/files*). Recuerda cerrar siempre los ficheros con el método `close()`. El siguiente ejemplo muestra cómo crear un fichero y escribir en él un texto:

```
String fichero = "fichero.txt";
String texto = "texto almacenado";
```

```
FileOutputStream fos;
try {
   fos = openFileOutput(fichero,Context.MODE_PRIVATE);
   fos.write(texto.getBytes());
   fos.close();
} catch (FileNotFoundException e) {
   Log.e("Mi Aplicación",e.getMessage(),e);
} catch (IOException e) {
   Log.e("Mi Aplicación",e.getMessage(),e);
}
```

Es muy importante hacer un manejo cuidadoso de los errores. De hecho, el acceso a ficheros ha de realizarse de forma obligatoria dentro de una sección `try/catch`.

Además de los dos métodos indicados, pueden serte útiles algunos de los siguientes: `getFilesDir()` devuelve la ruta absoluta donde se están guardando los ficheros; `getDir()` crea un directorio en tu almacenamiento interno (o lo abre si existe); `deleteFile()` borra un fichero; `fileList()` devuelve un *array* con los ficheros almacenados por tu aplicación.



Ejercicio: *Almacenando puntuaciones en un fichero de la memoria interna*

El siguiente ejercicio muestra una clase que implementa la interfaz `AlmacenPuntuaciones` utilizando los métodos antes descritos.

1. Abre el proyecto Asteroides.
2. Crea una nueva clase `AlmacenPuntuacionesFicheroInterno`.
3. Reemplaza el código por el siguiente:

```
public class AlmacenPuntuacionesFicheroInterno implements
AlmacenPuntuaciones {
   private static String FICHERO = "puntuaciones.txt";
   private Context context;

   public AlmacenPuntuacionesFicheroInterno(Context context) {
      this.context = context;
   }

   public void guardarPuntuacion(int puntos, String nombre, long fecha){
      try {
         FileOutputStream f = context.openFileOutput(FICHERO,
                                   Context.MODE_APPEND);
         String texto = puntos + " " + nombre + "\n";
         f.write(texto.getBytes());
         f.close();
      } catch (Exception e) {
         Log.e("Asteroides", e.getMessage(), e);
      }
   }
```

```
   public List<String> listaPuntuaciones(int cantidad) {
      List<String> result = new ArrayList<String>();
      try {
         FileInputStream f = context.openFileInput(FICHERO);
         BufferedReader entrada = new BufferedReader(
                              new InputStreamReader(f));
         int n = 0;
         String linea;
         do {
            linea = entrada.readLine();
            if (linea != null) {
               result.add(linea);
               n++;
            }
         } while (n < cantidad && linea != null);
         f.close();
      } catch (Exception e) {
         Log.e("Asteroides", e.getMessage(), e);
      }
      return result;
   }
}
```

4. Abre el fichero *MainActivity.java* y en el método `onCreate()` reemplaza la línea adecuada por:

```
almacen = new AlmacenPuntuacionesFicheroInterno(this);
```



Práctica: *Configurar almacenamiento de puntuaciones desde preferencias*

Modifica las preferencias de la aplicación Asteroides para que el usuario pueda seleccionar dónde se guardarán las puntuaciones. De momento incluye tres opciones: "Array", "Preferencias" y "Fichero en memoria interna".

1. Abre el fichero *MainActivity.java* y en el método `onCreate()` reemplaza:

```
almacen = new AlmacenPuntuacionesFicheroInterno(this);
```

   por el código necesario para que se inicialice la variable `almacen` de forma adecuada según el valor introducido en preferencias.

2. Verifica el resultado.

3. Observa que cuando desde las preferencias cambias de tipo de almacenamiento, no tiene efecto hasta que sales de la actividad principal y cargas de nuevo la aplicación. Para resolverlo, arranca la actividad de preferencias usando `startActivityForResult()`. En el método `onActivityResult()` verifica si se vuelve de esta actividad e inicializa de nuevo la variable `almacen`.

4. Verifica de nuevo el resultado.

5. Cada vez que añadas un nuevo método de almacenamiento inclúyelo en la lista de preferencias.



Preguntas de repaso: *Ficheros*

9.4.2. Sistema de almacenamiento externo

Los teléfonos Android suelen disponer de memoria adicional de almacenamiento, conocido como almacenamiento externo. Este almacenamiento suele ser de mayor capacidad, por lo que resulta ideal para almacenar ficheros de música o vídeo. Suele ser una memoria extraíble, como una tarjeta SD, o una memoria interna no extraíble (algunos modelos incorporan los dos tipos de memoria, es decir, almacenamiento externo extraíble y almacenamiento interno no extraíble). Cuando conectamos el dispositivo Android a través del cable USB permitimos el acceso a esta memoria externa, de forma que los ficheros aquí escritos podrán ser leídos, modificados o borrados por cualquier usuario.

Para acceder a la memoria externa, lo habitual es utilizar la ruta `/sdcard/...`

Esta es la carpeta donde el sistema monta la tarjeta SD. No obstante, resulta más conveniente utilizar el método `Environment.getExternalStorageDirectory()` para que el sistema nos indique la ruta exacta.

Resulta necesario solicitar el permiso `WRITE_EXTERNAL_STORAGE` en `AndroidManifest.xml` para poder escribir en la memoria externa. En la versión 4.1 aparece el permiso `READ_EXTERNAL_STORAGE`. Sin embargo, este permiso se ha introducido para un futuro uso. En la actualidad todas las aplicaciones pueden leer en la memoria externa. Por lo tanto, has de tener cuidado con la información que dejas en esta memoria.



Vídeo[tutorial]: *Almacenamiento externo en Android*



Ejercicio: *Almacenando puntuaciones en la memoria externa*

1. Abre el proyecto del ejercicio anterior.
2. Selecciona el fichero *AlmacenPuntuacionesFicheroInterno.java* y cópialo en el portapapeles (*Ctrl-C*).
3. Pega el fichero sobre el proyecto (*Ctrl-V*) y renómbralo como *AlmacenPuntuacionesFicheroExterno.java*.
4. Abre la nueva clase creada y reemplaza la inicialización de la variable `FICHERO` por:

```
    private static String FICHERO = Environment.
             getExternalStorageDirectory() + "/puntuaciones.txt";
```

Dependiendo de si utilizas un emulador o un dispositivo real, el valor de `FICHERO` será diferente. Posibles valores son: `"/sdcard/puntuaciones.txt"` o `"/storage/sdcard0/puntuaciones.txt"`.

5. En el método `guardarPuntuacion()` reemplaza la inicialización de `f` por:

```
FileOutputStream f = new FileOutputStream(FICHERO, true);
```

6. En el método `listaPuntuaciones()` reemplaza la inicialización de `f` por:

```
   FileInputStream f = new FileInputStream(FICHERO);
```

7. En el método `onCreate()` de la actividad `MainActivity` reemplaza la inicialización de `almacen` por:

```
almacen = new AlmacenPuntuacionesFicheroExterno(this);
```

O si has hecho la práctica *Configurar almacenamiento de puntuaciones desde preferencias* añade un nuevo tipo en las preferencias.

8. Abre el fichero *AndroidManifest.xml* y solicita el permiso `WRITE_EXTERNAL_STORAGE`. No es imprescindible que pidas permiso de lectura, dado que este ya está implícito cuando se pide el de escritura. Se trata de un permiso peligroso. Si ejecutas la aplicación en un dispositivo con versión 6 o superior, debes dar el permiso de forma manual desde los ajustes del terminal.
9. Ejecuta la aplicación y crea nuevas puntuaciones.
10. Verifica con la vista *File Explorer* que en la carpeta *sdcard* aparece el fichero.



Práctica: *Solicitar permiso de acceso a memoria externa*

Antes de leer o escribir el fichero en la memoria externa, comprueba que tienes permiso. En caso negativo, solicítalo al usuario. Puedes basarte en el ejercicio: *Solicitud de permisos en Android Marshmallow*.

Verificando acceso a la memoria externa

La memoria externa puede haber sido extraída o estar protegida contra escritura. Puedes utilizar el método `Environment.getExternalStorageState()` para verificar el estado de la memoria. Veamos cómo se utiliza:

```
String stadoSD = Environment.getExternalStorageState();
if (stadoSD.equals(Environment.MEDIA_MOUNTED)) {
    // Podemos leer y escribir
    ...
} else if (stadoSD.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
    // Podemos leer
    ...
```

```
} else {
    // No podemos leer y ni escribir
    ...
}
```



Práctica: *Verificando acceso a la memoria externa*

1. Modifica la clase `AlmacenPuntuacionesFicheroExterno` para que antes de acceder a la memoria externa verifique que la operación es posible. En caso contrario mostrará un `Toast` y saldrá del método.

2. Ejecuta el programa en un dispositivo real con memoria externa y verifica que se almacena correctamente.

3. Ahora verifica el comportamiento cuando la memoria externa no está disponible. Para que el dispositivo ya no tenga acceso a esta memoria, la solución más sencilla consiste en conectar el dispositivo con el cable USB y activar el almacenamiento por USB.



Solución: *Verificando acceso a la memoria externa*

1. En `guardarPuntuacion()` añade:

```
String stadoSD = Environment.getExternalStorageState();
if (!stadoSD.equals(Environment.MEDIA_MOUNTED)) {
   Toast.makeText(context, "No puedo escribir en la memoria externa",
                                              Toast.LENGTH_LONG).show();
   return;
}
```

2. En `listaPuntuaciones()` añade:

```
String stadoSD = Environment.getExternalStorageState();
if (!stadoSD.equals(Environment.MEDIA_MOUNTED) &&
          !stadoSD.equals(Environment.MEDIA_MOUNTED_READ_ONLY)) {
   Toast.makeText(context, "No puedo leer en la memoria externa",
                                          Toast.LENGTH_LONG).show();
   return result;
}
```

Almacenando ficheros específicos de tu aplicación en el almacenamiento externo

Las aplicaciones pueden almacenar los ficheros en una carpeta específica del sistema de almacenamiento externo, de forma que cuando la aplicación sea desinstalada se borren automáticamente estos ficheros. En concreto, esta carpeta ha de seguir esta estructura:

```
/Android/data/<nombre_del_paquete>/files/
```

Donde el paquete *<nombre_del_paquete>* ha de cambiarse por el nombre del paquete de la aplicación, por ejemplo `org.example.asteroides`.

Una gran ventaja de trabajar en este almacenamiento es que a partir del API 19 (v4.4) no es necesario pedir permiso de almacenamiento.

Puedes utilizar el método `getExternalFilesDir(null)` para obtener esta ruta. Si en lugar de `null` indicas alguna de las constantes que se indican más abajo, se devolverá la ruta a una carpeta específica según el tipo de contenido que nos interese. Este método crea la carpeta en caso de no existir previamente. Indicando la carpeta garantizamos que el escáner de medios de Android categoriza los ficheros de forma adecuada. Por ejemplo, un tono de llamada será identificado como tal y no como un fichero de música. De esta forma, no aparecerá en la lista de música que puede reproducir el reproductor multimedia. Estas carpetas también son eliminadas cuando se desinstala la aplicación.

Constante	Carpeta	Descripción
`DIRECTORY_MUSIC`	`Music`	Ficheros de música
`DIRECTORY_PODCASTS`	`Podcasts`	Descargas desde podcast
`DIRECTORY_RINGTONES`	`Ringtones`	Tono de llamada de teléfono
`DIRECTORY_ALARMS`	`Alarms`	Sonidos de alarma
`DIRECTORY_NOTIFICATIONS`	`Notifications`	Sonidos para notificaciones
`DIRECTORY_PICTURES`	`Pictures`	Ficheros con fotografías
`DIRECTORY_DOWNLOADS`	`Download`	Descargas de cualquier tipo
`DIRECTORY_DCIM`	`DCIM`	Carpeta que tradicionalmente crean las cámaras

Práctica: *Almacenando puntuaciones en una carpeta de la aplicación de la memoria externa*

1. Selecciona el fichero *AlmacenPuntuacionesFicheroExterno.java*, y cópialo en el portapapeles (*Ctrl-C*).
2. Pega el fichero sobre el proyecto (*Ctrl-V*) y renómbralo como *AlmacenPuntuacionesFicheroExtApl.java*.
3. Modifica los métodos `listaPuntuaciones()` y `guardarPuntuacion()` para que las puntuaciones se almacenen en la memoria externa, pero en una carpeta de tu aplicación.
4. Modifica el código correspondiente para que la nueva clase pueda ser seleccionada como almacén de las puntuaciones.
5. Ejecuta la aplicación. Desinstala la aplicación y verifica si el fichero ha sido eliminado.

Almacenando ficheros compartidos en el almacenamiento externo

Si quieres crear un fichero que no sea específico para tu aplicación y quieres que no sea borrado cuando tu aplicación sea desinstalada, puedes crearlo en cualquier otro directorio del almacenamiento externo.

Lo ideal es que utilices alguno de los directorios públicos creados para almacenar diferentes tipos de ficheros. Estos directorios parten de la raíz del almacenamiento externo y siguen con alguna de las carpetas listadas en la tabla anterior.

A partir del nivel de API 8 puedes utilizar el método `getExternalStoragePublicDirectory(String tipo)` para obtener esta ruta de uno de estos directorios compartidos. Como parámetro utiliza alguna de las constantes que se indican en la tabla anterior. Guardando los ficheros en las carpetas adecuadas garantizamos que el escáner de medios de Android categoriza los ficheros de forma adecuada. Si utilizas un nivel de API anterior al 8, lo recomendable es crear estas carpetas manualmente.

> ***NOTA:*** *Si quieres que tus ficheros estén ocultos al escáner de medios, incluye un fichero vacío que se llame .nomedia en la carpeta donde estén almacenados.*

Almacenamiento externo con varias unidades

Algunos dispositivos incluyen varias unidades de almacenamiento externo. En este caso, al conectar el dispositivo con un cable USB a un ordenador aparecerá más de una unidad:



En estos casos, una unidad suele corresponder a una tarjeta extraíble SD y la otra una partición en la memoria *flash*. Si utilizamos el método `getExternalFilesDir()`, y los relacionados, nos devolverá una de las unidades. Esta unidad se denomina unidad de almacenamiento primaria y el resto de unidades, secundarias. Es el fabricante quien decide cuál de las unidades es la memoria primaria. Normalmente Samsung escoge como memoria externa primaria la partición *flash* no extraíbe.

Hasta la versión 4.4 el API de Android no soportaba múltiples unidades de memoria externa. Solo podíamos acceder de forma estándar a la memoria externa primaria y para acceder a la memoria externa secundaria es necesario conocer dónde el fabricante ha montado esta memoria. En la mayoría de los casos se monta en */mnt/sdcard/external_sd.*

A partir de la versión 4.4 se incorporan varios métodos que nos permiten trabajar con varias unidades externas. En la clase `Context` se añade `File[] getExternalFilesDirs(String)`, que nos devuelve un array con la ruta a cada uno de los almacenamientos externos disponibles. El primer elemento ha de coincidir con la ruta devuelta por `getExternalFilesDir(String)`. La clase `Environment` incorpora el método estático `String getStorageState(File)`, que permite conocer

el estado de cada unidad de almacenamiento. Nos devuelve una información equivalente a la del método `getExternalStorageState()`.



Desafio: Permitir seleccionar diferentes almacenamientos externos

En caso de existir varios almacenamientos externos, se mostrará al usuario un listado para que seleccione dónde se almacenarán las puntuaciones. Solo podrá realizarse en versiones superiores a la 4.4.



Preguntas de repaso: *La memoria externa*

9.4.3. Acceder a un fichero de los recursos

También tienes la posibilidad de almacenar ficheros en los recursos, es decir, adjuntos al paquete de la aplicación. Has de tener en cuenta que estos ficheros no podrán ser modificados.

Tienes dos alternativas para esto: usar la carpeta `res/raw` o `assets`. La principal diferencia a la hora de usar una carpeta u otra está en la forma de identificar el fichero. Por ejemplo, si arrastras un fichero que se llame `datos.txt` a la carpeta `res/raw`, podrás acceder a él usando `context.getResources.openRawResource(R.raw.datos)`. Si, por el contrario, dejas este fichero en la carpeta `assets`, podrás acceder a él usando `context.getAssets(). open("datos.txt")`. Otra diferencia es que dentro de `assets` podrás crear subcarpetas para organizar los ficheros.

Recuerda que, tanto en la carpeta `raw` como en `assets`, los ficheros nunca son comprimidos.



Ejercicio: *Leyendo puntuaciones de un fichero de recursos en* res/raw

1. Con el explorador de ficheros busca en el terminal un fichero de texto que se llame *puntuaciones.txt*, creado en alguno de los ejercicios anteriores.
2. Extráelo del terminal y pégalo en la carpeta *res/raw* del proyecto Asteroides.
3. Selecciona el fichero *AlmacenPuntuacionesFicheroInterno.java* y cópialo en el portapapeles (*Ctrl-C*).
4. Pega el fichero sobre el proyecto (*Ctrl-V*) y renómbralo como *AlmacenPuntuacionesRecursoRaw.java*.
5. Elimina de esta clase todo el código del método `guardarPuntuacion()`. No se realiza ninguna acción en este método.

6. Para que las puntuaciones se lean del fichero de los recursos, en el método `listaPuntuaciones()` reemplaza:

```
FileInputStream f = context.openFileInput(FICHERO);
```

por:

```
InputStream f = context.getResources().openRawResource(
                                        R.raw.puntuaciones);
```

7. La siguiente línea ya no tiene sentido. Elimínala:

```
private static String FICHERO = "puntuaciones.txt";
```

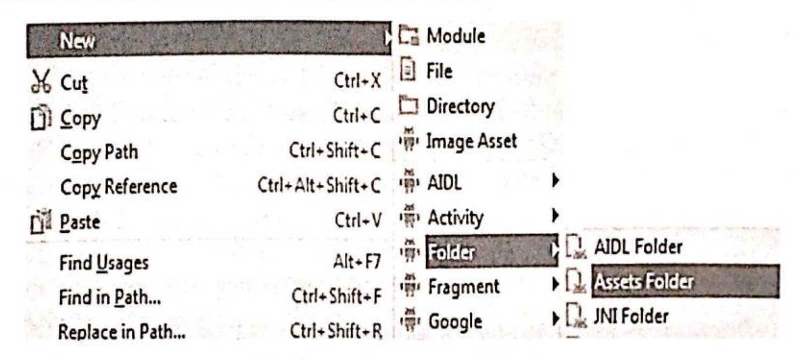
8. Modifica el código correspondiente para que la nueva clase pueda ser seleccionada como almacén de las puntuaciones.

9. Verifica el resultado.

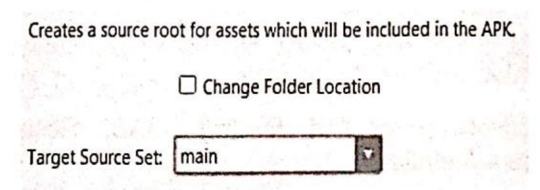


Ejercicio: *Leyendo puntuaciones de un fichero de recursos en assets*

1. Selecciona *File / New / Folder / Assets Folder.*



En la siguiente ventana deja los valores por defecto:



Hemos creado la carpeta *assets* que aparecerá dentro de *res*. Vamos a crear la subcarpeta *carpeta* dentro de esta carpeta. Pulsa con el botón derecho sobre *assets*, selecciona *New/Directory* e introduce "carpeta".

2. Copia el fichero *puntuaciones.txt* dentro de la carpeta que acabas de crear.

3. Selecciona el fichero *AlmacenPuntuacionesRecursoRaw.java* y cópialo en el portapapeles (*Ctrl-C*).

4. Pega el fichero sobre el proyecto (*Ctrl-V*) y renómbralo como *AlmacenPuntuacionesRecursoAssets.java*.

5. En el método `listaPuntuaciones()` reemplaza:

```
InputStream f = context.getResources().openRawResource(
                                        R.raw.puntuaciones);
```

por:

```
InputStream f = context.getAssets().open("carpeta/puntuaciones.txt");
```

6. Modifica el código correspondiente para que la nueva clase pueda ser seleccionada como almacén de las puntuaciones.

7. Verifica que el resultado es idéntico al ejercicio anterior.

9.5. Trabajando con XML

Como sabrás, XML es uno de los estándares más utilizados en la actualidad para codificar información. Es ampliamente utilizado en Internet; además, como hemos mostrado a lo largo de este libro, se utiliza para múltiples usos en el SDK de Android. Entre otras cosas, es utilizado para definir *layouts*, animaciones, *AndroidManifest.xml*, etc.

Una de las mayores fortalezas de la plataforma Android es que se aprovecha el lenguaje de programación Java y sus librerías. El SDK de Android no acaba de ofrecer todo lo disponible para su estándar del entorno de ejecución Java (JRE), pero es compatible con una fracción muy significativa de este. Lo mismo ocurre en lo referente a trabajar con XML: Java dispone de una gran cantidad de API con este propósito, pero no todas están disponibles desde Android.

Librerías disponibles:

Java's Simple API for XML (SAX) (paquetes `org.xml.sax.*`).

Document Object Model (DOM) (paquetes `org.w3c.dom.*`).

Librerías no disponibles:

Streaming API for XML (StAX). Aunque se dispone de otra librería con funcionalidad equivalente (paquete `org.xmlpull.v1.XmlPullParser`).

Java Architecture for XML Binding (JAXB). Resultaría demasiado pesada para Android.

Como podrás ver al estudiar los ejemplos, leer y escribir ficheros XML es muy laborioso y necesitarás algo de esfuerzo para comprender el código empleado. Vamos a explicar las dos alternativas más importantes, SAX y DOM. El planteamiento es bastante diferente. Tras ver los ejemplos podrás decidir qué herramienta se adapta mejor a tus gustos personales o al problema en concreto que tengas que resolver.

El ejemplo utilizado para ilustrar el trabajo con XML será el mismo que el utilizado en el resto del capítulo: almacenar las mejores puntuaciones obtenidas. El formato XML que se utilizará para este propósito se muestra a continuación:

```
<?xml version="1.0" encoding="UTF-8"?>
<lista_puntuaciones>
    <puntuacion fecha="1288122023410">
        <nombre>Mi nombre</nombre>
        <puntos>45000</puntos>
    </puntuacion>
    <puntuacion fecha="1288122428132">
        <nombre>Otro nombre</nombre>
        <puntos>31000</puntos>
    </puntuacion>
</lista_puntuaciones>
```

9.5.1. Procesando XML con SAX

El uso de la API SAX (*Simple API for XML*) se recomienda cuando se desea un programa de análisis rápido y se quiere reducir al mínimo el consumo de memoria de la aplicación. Eso hace que sea muy apropiada para un dispositivo móvil con Android. También resulta ventajosa para procesar ficheros de gran tamaño.

SAX nos facilita realizar un *parser* (analizador) sobre un documento XML para así poder analizar su estructura. Ha de quedar claro que SAX no almacena los datos. Por lo tanto, necesitaremos una estructura de datos donde guardar la información contenida en el XML. Para realizar este *parser* se generarán una serie de eventos a medida que se vaya leyendo el documento secuencialmente. Por ejemplo, al analizar el documento XML anterior, SAX generará los siguientes eventos:

```
Comienza elemento: lista_puntuaciones
Comienza elemento: puntuacion, con atributo fecha="1288122023410"
Comienza elemento: nombre
Texto de nodo: Mi nombre
Finaliza elemento: nombre
Comienza elemento: puntos
Texto de nodo: 45000
Finaliza elemento: puntos
Finaliza elemento: puntuacion
Comienza elemento: puntuacion, con atributo fecha="1288122428132"
Comienza elemento: nombre
Texto de nodo: Otro nombre
Finaliza elemento: nombre
Comienza elemento: puntos
Texto de nodo: 31000
Finaliza elemento: puntos
Finaliza elemento: puntuacion
Finaliza elemento: lista_puntuaciones
```

Para analizar un documento mediante SAX, vamos a escribir métodos asociados a cada tipo de evento. Este proceso se realiza extendiendo la clase `DefaultHandler`, que nos permite rescribir 5 métodos. Los métodos listados a continuación serán llamados a medida que ocurran los eventos listados anteriormente.

`startDocument()`: Comienza el documento XML.

`endDocument()`: Finaliza documento XML.

`startElement(String uri, String nombreLocal, String nombreCualif, Attributes atributos)`: Comienza una nueva etiqueta; se indican los parámetros:

`uri`: La uri del espacio de nombres o vacío, si no se ha definido.

`nombreLocal`: Nombre local de la etiqueta sin prefijo.

`nombreCualif`: Nombre cualificado de la etiqueta con prefijo.

`atributos`: Lista de atributos de la etiqueta.

`endElement(String uri, String nombreLocal, String nombreCualif)`: Termina una etiqueta.

`characters(char ch[], int comienzo, int longitud)`: Devuelve en `ch` los caracteres dentro de una etiqueta. Es decir, en `<etiqueta> caracteres </etiqueta>` devolvería `caracteres`. Para obtener un `String` con estos caracteres: `String s = new String(ch,comienzo,longitud)`. Más adelante veremos un ejemplo de cómo utilizar este método.



Ejercicio: *Almacenando puntuaciones en XML con SAX*

Una vez descritos los principios de trabajo con SAX, pasemos a implementar la interfaz `AlmacenPuntuaciones` mediante esta API.

1. Crea la clase `AlmacenPuntuacionesXML_SAX` en la aplicación `Asteroides` y escribe el siguiente código:

```
public class AlmacenPuntuacionesXML_SAX implements AlmacenPuntuaciones {
   private static String FICHERO = "puntuaciones.xml";
   private Context contexto;
   private ListaPuntuaciones lista;
   private boolean cargadaLista;

   public AlmacenPuntuacionesXML_SAX(Context contexto) {
      this.contexto = contexto;
      lista = new ListaPuntuaciones();
      cargadaLista = false;
   }

   @Override
   public void guardarPuntuacion(int puntos, String nombre, long fecha) {
      try {
         if (!cargadaLista){
            lista.leerXML(contexto.openFileInput(FICHERO));
         }
      } catch (FileNotFoundException e) {
      } catch (Exception e) {
         Log.e("Asteroides", e.getMessage(), e);
      }
```

```
      lista.nuevo(puntos, nombre, fecha);
      try {
         lista.escribirXML(contexto.openFileOutput(FICHERO,
                                              Context.MODE_PRIVATE));
      } catch (Exception e) {
         Log.e("Asteroides", e.getMessage(), e);
      }
   }
   @Override
   public List<String> listaPuntuaciones(int cantidad) {
      try {
         if (!cargadaLista){
            lista.leerXML(contexto.openFileInput(FICHERO));
         }
      } catch (Exception e) {
         Log.e("Asteroides", e.getMessage(), e);
      }
      return lista.aListString();
   }
```

La nueva clase comienza definiendo una serie de variables y constantes. En primer lugar, el nombre del fichero donde se guardarán los datos. Con el valor indicado, el fichero se almacenará en `/data/data/org.example.asteroides/files/puntuaciones.xml`. Pero puedes almacenarlos en otro lugar, como por ejemplo en la memoria SD. La variable más importante es `lista` de la clase `ListaPuntuaciones`. En ella guardaremos la información contenida en el fichero XML. Esta clase se define a continuación. La variable `cargadaLista` nos indica si `lista` ya ha sido leída desde el fichero.

El código continúa sobrescribiendo los dos métodos de la interfaz. En `guardarPuntuacion()` comenzamos verificando si `lista` ya ha sido cargada, para hacerlo en caso necesario. Es posible que el programa se esté ejecutando por primera vez, en cuyo caso el fichero no existirá. En este caso se producirá una excepción de tipo `FileNotFoundException` al tratar de abrir el fichero. Esta excepción es capturada por nuestro código, pero no realizamos ninguna acción dado que no se trata de un verdadero error. A continuación se añade un nuevo elemento a `lista` y se escribe de nuevo el fichero XML. El siguiente método, `listaPuntuacion()`, resulta sencillo de entender, al limitarse a métodos definidos en la clase `ListaPuntuaciones`.

2. Pasemos a mostrar el comienzo de la clase `ListaPuntuaciones`. No es necesario almacenarla en un fichero aparte, puedes definirla dentro de la clase anterior. Para ello copia el siguiente código justo antes del último `}` de la clase `AlmacenPuntuacionesXML_SAX`:

```
private class ListaPuntuaciones {

   private class Puntuacion {
      int puntos;
      String nombre;
      long fecha;
   }
```

```
    private List<Puntuacion> listaPuntuaciones;

    public ListaPuntuaciones() {
       listaPuntuaciones = new ArrayList<Puntuacion>();
    }

    public void nuevo(int puntos, String nombre, long fecha) {
       Puntuacion puntuacion = new Puntuacion();
       puntuacion.puntos = puntos;
       puntuacion.nombre = nombre;
       puntuacion.fecha = fecha;
       listaPuntuaciones.add(puntuacion);
    }

    public List <String> aListString() {
       List<String> result = new ArrayList<String>();
       for (Puntuacion puntuacion : listaPuntuaciones) {
          result.add(puntuacion.nombre+" "+puntuacion.puntos);
       }
       return result;
    }
```

El objetivo de esta clase es mantener una lista de objetos `Puntuacion`. Dispone de métodos para insertar un nuevo elemento (`nuevo()`) y devolver una lista con todas las puntuaciones almacenadas (`aListString()`).

3. Lo verdaderamente interesante de esta clase es que permite la lectura y escritura de los datos desde un documento XML (`leerXML()` y `escribirXML()`). Veamos primero cómo leer un documento XML usando SAX. Escribe el siguiente código a continuación del anterior:

```
public void leerXML(InputStream entrada) throws Exception {
    SAXParserFactory fabrica = SAXParserFactory.newInstance();
    SAXParser parser = fabrica.newSAXParser();
    XMLReader lector = parser.getXMLReader();
    ManejadorXML manejadorXML = new ManejadorXML();
    lector.setContentHandler(manejadorXML);
    lector.parse(new InputSource(entrada));
    cargadaLista = true;
}
```

Para leer un documento XML comenzamos creando una instancia de la clase `SAXParserFactory`, lo que nos permite crear un nuevo *parser* XML de tipo `SAXParser`. Luego creamos un lector, de la clase `XMLReader`, asociado a este *parser*. Creamos `manejadorXML` de la clase `ManejadorXML` y asociamos este manejador al `XMLReader`. Para finalizar, le indicamos al `XMLReader` qué entrada tiene para que realice el proceso de *parser*. Una vez finalizado el proceso, marcamos que el fichero está cargado.

Como ves, el proceso es algo largo, pero siempre se realiza igual. Donde sí que tendremos que trabajar algo más es en la creación de la clase `ManejadorXML`, dado que va a depender del formato del fichero que queramos leer. Esta clase se lista en el siguiente punto.

4. Escribe este código a continuación del anterior:

```
class ManejadorXML extends DefaultHandler {
    private StringBuilder cadena;
    private Puntuacion puntuacion;

    @Override
    public void startDocument() throws SAXException {
        listaPuntuaciones = new ArrayList<Puntuacion>();
        cadena = new StringBuilder();
    }

    @Override
    public void startElement(String uri, String nombreLocal, String
                nombreCualif, Attributes atr) throws SAXException {
        cadena.setLength(0);
        if (nombreLocal.equals("puntuacion")) {
            puntuacion = new Puntuacion();
            puntuacion.fecha = Long.parseLong(atr.getValue("fecha"));
            }
    }

    @Override
    public void characters(char ch[], int comienzo, int lon) {
        cadena.append(ch, comienzo, lon);
    }

    @Override
    public void endElement(String uri, String nombreLocal,
                String nombreCualif) throws SAXException {
        if (nombreLocal.equals("puntos")) {
            puntuacion.puntos = Integer.parseInt(cadena.toString());
        } else if (nombreLocal.equals("nombre")) {
            puntuacion.nombre = cadena.toString();
        } else if (nombreLocal.equals("puntuacion")) {
            listaPuntuaciones.add(puntuacion);
        }
    }

    @Override
    public void endDocument() throws SAXException {}
}
```

Esta clase define un manejador que captura los cinco eventos generados en el proceso de *parsing* en SAX. En `startDocument()` nos limitamos a inicializar variables. En `startElement()` verificamos que hemos llegado a una etiqueta `<puntuación>`. En tal caso, creamos un nuevo objeto de la clase `Puntuacion` e inicializamos el campo `fecha` con el valor indicado en uno de los atributos.

El método `characters()` se llama cuando aparece texto dentro de una etiqueta (`<etiqueta> caracteres </etiqueta>`). Nos limitamos a almacenar este texto en la variable `cadena` para utilizarlo en el siguiente método. SAX no nos garantiza que nos pasará todo el texto en un solo evento: si el texto es muy extenso, se realizarán varias llamadas a este método. Por esta razón, el texto se va acumulando en `cadena`.

El método endElement() resulta más complejo, dado que en función de que etiqueta esté acabando realizaremos una tarea diferente. Si se trata de </puntos> o de </nombre> utilizaremos el valor de la variable cadena para actualizar el valor correspondiente. Si se trata de </puntuacion> añadimos el objeto puntuacion a la lista.

5. Introduce a continuación el último método de la clase ListaPuntuaciones, que nos permite escribir el documento XML:

```
   public void escribirXML(OutputStream salida) {
      XmlSerializer serializador = Xml.newSerializer();
      try {
         serializador.setOutput(salida, "UTF-8");
         serializador.startDocument("UTF-8", true);
         serializador.startTag("", "lista_puntuaciones");
         for (Puntuacion puntuacion : listaPuntuaciones) {
            serializador.startTag("", "puntuacion");
            serializador.attribute("", "fecha",
                              String.valueOf(puntuacion.fecha));
            serializador.startTag("", "nombre");
            serializador.text(puntuacion.nombre);
            serializador.endTag("", "nombre");
            serializador.startTag("", "puntos");
            serializador.text(String.valueOf(puntuacion.puntos));
            serializador.endTag("", "puntos");
            serializador.endTag("", "puntuacion");
         }
         serializador.endTag("", "lista_puntuaciones");
         serializador.endDocument();
      } catch (Exception e) {
         Log.e("Asteroides", e.getMessage(), e);
      }
   }
} //Cerramos ListaPuntuaciones
} //Cerramos AlmacenPuntuacionesXML_SAX
```

Como puedes ver, todo el trabajo se realiza por medio de un objeto de la clase XmlSerializer, que escribe el código XML en el OutputStream que hemos pasado como parámetros.

6. La variable almacen ha de inicializarse de forma adecuada.

7. Modifica el código correspondiente para que este método pueda ser seleccionado para almacenar las puntuaciones.

8. Verifica el resultado.

9.5.2. Procesando XML con DOM

DOM (*Document Object Model*) es una API creada por W3C (World Wide Web Consortium) que nos permite manipular dinámicamente documentos XML y HTML. Android soporta el nivel de especificación 3, por lo que permite trabajar con definición de tipo de documento (DTD) y validación de documentos.

Como ya hemos comentado, el planteamiento de DOM es muy diferente del de SAX. SAX recorre todo el documento XML secuencialmente y lo analiza, pero sin almacenarlo. Por el contrario, DOM permite cargar el documento XML en memoria RAM y manipularlo directamente en memoria. DOM representa el documento como un árbol. Podremos crear nuevos nodos, borrar o modificar los existentes. Una vez dispongamos de la nueva versión, podremos almacenarlo en un fichero o mandarlo por Internet.

Trabajar con DOM tiene sus ventajas frente a SAX: por ejemplo, nos evitamos definir a mano el proceso de *parser* y crear una estructura para almacenar los datos. Pero también tiene sus inconvenientes: recorrer un documento DOM puede ser algo complejo; además, al tener que cargarse todo el documento en memoria puede consumir excesivos recursos para un dispositivo como un teléfono móvil. Este inconveniente cobra especial relevancia al trabajar con documentos grandes. Para terminar, DOM procesa la información de forma más lenta.



Enlaces de interés: Procesado XML con DOM

http://www.androidcurso.com/index.php/818



Preguntas de repaso: *Trabajando con XML*

9.6. Trabajando con JSON

JSON corresponde al acrónimo de *JavaScript Object Notation*. Es un formato para representar información similar a XML, pero presenta dos ventajas frente a este: es más compacto, pues necesita menos bytes para codificar la información y el código necesario para realizar un parser es mucho menor. Estas ventajas hacen que cada vez sea más popular, especialmente en el intercambio de datos a través de la red. A continuación, se muestra cómo se codificaría el ejemplo que estamos desarrollando en este capítulo:

```
{
  "puntuaciones": [
    { "fecha": 1288122023410, "nombre": "Mi nombre", "puntos": 45000 },
    { "fecha": 1288122428132, "nombre": "Otro nombre", "puntos": 31000 }
  ]
}
```

Comparando el número de caracteres empleados frente al que se utilizó para codificar esta información en XML, podemos observar una reducción cercana al 50 %.

La plataforma Android incorpora la librería estándar `org.json` con la que podremos procesar ficheros JSON. Otra alternativa es la librería `com.google.gson` que va a resultar más sencilla de utilizar. Veamos estas dos alternativas.

9.6.1. Procesando JSON con la librería Gson

GSON es una librería de código abierto creada por Google que permite serializar objetos Java para convertirlos en un `String`. Su uso más frecuente es para convertir un objeto en su representación JSON y a la inversa.

La gran ventaja de esta librería es que puede ser usada sobre objetos de cualquier tipo de clases, incluso clases preexistentes que no has creado. Esto es posible al no ser necesario introducir código en las clases para que sean serializadas.

El código necesario es muy reducido, como se muestra a continuación:

```
private ArrayList<Puntuacion> puntuaciones=new ArrayList<>();
private Gson gson = new Gson();
private Type type = new TypeToken<List<Puntuacion>>() {}.getType();

// Convertimos la colección de datos a un String JSON
String string = gson.toJson(puntuaciones, type);

// Convertimos un String JSON a una la colección de datos
puntuaciones = gson.fromJson(string, type);
```

En este código, `puntuaciones` contiene una colección (List) de elementos de tipo `Puntuacion`. Para representar estos datos en JSON vamos a necesitar un objeto `Gson` y otro `Type`. Este último representa el tipo de datos con el que trabajamos. En la variable `string` se almacenará el contenido de `puntuaciones` en representación JSON. En la siguiente línea se hace el proceso inverso.

La librería no solo permite transformar los datos en JSON, también podemos personalizar la serialización de los datos según las necesidades del programador. También permite excluir algunos atributos para que sean incluidos en la representación JSON.



Ejercicio: *Guardar puntuaciones en JSON con la librería Gson*

1. Crea la clase `Puntuacion` con el siguiente código:

```
public class Puntuacion {
    private int puntos;
    private String nombre;
    private long fecha;

    public Puntuacion(int puntos, String nombre, long fecha) {
        this.puntos = puntos;
        this.nombre = nombre;
        this.fecha = fecha;
    }
}
```

2. Sitúate al final de la clase y selecciona *Code > Generate > Getter and Setter*. Selecciona todos los atributos y pulsa *OK*.

3. Añade al fichero Gradle *Scripts/Bulid.gradle (Module:app)* la dependencia:

```
dependencies {
    ...
    implementation 'com.google.code.gson:gson:2.8.5'
}
```

4. Crea la clase `AlmacenPuntuacionesGSon` con el siguiente código:

```
public class AlmacenPuntuacionesGSon implements AlmacenPuntuaciones {
   private String string; //Almacena puntuaciones en formato JSON
   private Gson gson = new Gson();
   private Type type = new TypeToken<List<Puntuacion>>() {}.getType();

   public AlmacenPuntuacionesGSon() {
      guardarPuntuacion(45000,"Mi nombre", System.currentTimeMillis());
      guardarPuntuacion(31000,"Otro nombre", System.currentTimeMillis());
   }

   @Override
   public void guardarPuntuacion(int puntos, String nombre, long fecha) {
      //string = leerString();
      ArrayList<Puntuacion> puntuaciones;
      if (string == null) {
         puntuaciones = new ArrayList<>();
      } else {
         puntuaciones = gson.fromJson(string, type);
      }
      puntuaciones.add(new Puntuacion(puntos, nombre, fecha));
      string = gson.toJson(puntuaciones, type);
      //guardarString(string);
   }

   @Override
   public List<String> listaPuntuaciones(int cantidad) {
      //string = leerString();
      ArrayList<Puntuacion> puntuaciones;
      if (string == null) {
         puntuaciones = new ArrayList<>();
      } else {
         puntuaciones = gson.fromJson(string, type);
      }
      List<String> salida = new ArrayList<>();
      for (Puntuacion puntuacion : puntuaciones) {
         salida.add(puntuacion.getPuntos()+" "+puntuacion.getNombre());
      }
      return salida;
   }
}
```

En la variable `string` se almacenará la lista de puntuaciones en representación JSON. Para que los datos se almacenen de forma no volátil tendrías que implementar los métodos `guardarString()` y `leerString()`. La forma más sencilla sería almacenarlo en un fichero de preferencias. Otra alternativa sería guardarlo en un fichero en la memoria interna o externa. En el próximo capítulo veremos cómo mandar este `String` a través de Internet.

5. Modifica el código correspondiente para que se pueda seleccionar esta clase para el almacenamiento.

6. Ejecuta el proyecto y verifica su funcionamiento. Si visualizas el valor de `string` este debe ser:

```
[{"fecha":1478552190154,"nombre":"Mi nombre", "puntos":45000},
 {"fecha":1478552205944,"nombre":"Otro nombre","puntos":31000}]
```

NOTA: Observa como los atributos son almacenados por orden alfabético.



Práctica: *Guardar el string JSON en un fichero.*

1. Implementa los métodos `guardarString(String)` y `leerString()` para que la información se almacene en un fichero de preferencias o en la memoria del dispositivo.

2. En la versión anterior, la variable `string` hacía el papel de almacén de la información. En la nueva versión, este papel ha pasado a un fichero o a una preferencia. Elimina la variable global `string` y conviértela en variable local en los métodos donde sea necesario.



Ejercicio: *Guardar una clase en JSON con la librería Gson.*

El resultado del ejercicio anterior es muy similar al ejemplo JSON, mostrado al principio de este apartado. Sin embargo, no es exactamente igual. En el ejemplo se muestra un objeto JSON que incluye una única propiedad con nombre "puntuaciones". En el siguiente ejercicio veremos cómo obtener una estructura como esta.

1. En `AlmacenPuntuacionesGSon` añade la siguiente clase:

```
public class Clase {
   private ArrayList<Puntuacion> puntuaciones = new ArrayList<>();
   private boolean guardado;
}
```

2. Reemplaza el código subrayado de los siguientes métodos:

```
private Type type = new TypeToken<Clase>() {}.getType();

@Override
public void guardarPuntuacion(int puntos, String nombre, long fecha) {
   //string = leerString();
   Clase objeto;
   if (string == null) {
      objeto = new Clase();
   } else {
      objeto = gson.fromJson(string, type);
   }
```

```
    objeto.puntuaciones.add(new Puntuacion(puntos, nombre, fecha));
    string = gson.toJson(objeto, type);
    //guardarString(string);
}

@Override
public List<String> listaPuntuaciones(int cantidad) {
    //string = leerString();
    Clase objeto;
    if (string == null) {
       objeto = new Clase();
    } else {
       objeto = gson.fromJson(string, type);
    }
    List<String> salida = new ArrayList<>();
    for (Puntuacion puntuacion : objeto.puntuaciones) {
       salida.add(puntuacion.getPuntos()+" "+puntuacion.getNombre());
    }
    return salida;
}
```

3. Ejecuta el proyecto y verifica su funcionamiento. Si visualizas el valor de `string` este ha de ser:


```
{"guardado":false,
 "puntuaciones":[{"fecha":1478552190154,"nombre":"Mi nombre", "puntos":45000},
                 {"fecha":1478552205944,"nombre":"Otro nombre","puntos":31000}
                ]
}
```


Los saltos de línea han sido introducidos para facilitar la visualización.

Enlace de interés: Si te pasan un fichero JSON puede ser complejo crear el POJO adecuado para leerlo, especialmente si tiene objetos dentro de objetos, incluso con varios niveles de anidación. Este proceso puede realizarse de forma automática usando la herramienta http://www.jsonschema2pojo.org/

9.6.2. Procesando JSON con la librería org.json

La librería `org.json` permite tanto codificar datos en formato JSON dentro de un `String`, como el proceso inverso. Una de sus ventajas es que esta librería ya se encuentra integrada en la plataforma Android.

Para trabajar con esta librería hay que realizar el proceso de conversión manualmente, insertando cada elemento de uno en uno. Esto puede darnos más trabajo que otras librerías como GSON pero, al no ser un proceso automático, vamos a poder realizarlo de forma personalizada. Por ejemplo, podremos elegir el orden en que se generan los datos.



Ejercicio: *Guardar puntuaciones en JSON con la librería org.json.*

1. Crea la clase `AlmacenPuntuacionesJSon` con el siguiente código:

```
public class AlmacenPuntuacionesJSon implements AlmacenPuntuaciones {
   private String string; //Almacena puntuaciones en formato JSON

   public AlmacenPuntuacionesJSon() {
      guardarPuntuacion(45000,"Mi nombre", System.currentTimeMillis());
      guardarPuntuacion(31000,"Otro nombre", System.currentTimeMillis());
   }
   @Override
   public void guardarPuntuacion(int puntos, String nombre, long fecha) {
      //string = leerString();
      List<Puntuacion> puntuaciones = leerJSon(string);
      puntuaciones.add(new Puntuacion(puntos, nombre, fecha));
      string = guardarJSon(puntuaciones);
      //guardarString(string);
   }

   @Override
   public List<String> listaPuntuaciones(int cantidad) {
      //string = leerFichero();
      List<Puntuacion> puntuaciones = leerJSon(string);
      List<String> salida = new ArrayList<>();
      for (Puntuacion puntuacion: puntuaciones) {
         salida.add(puntuacion.getPuntos()+" "+puntuacion.getNombre());
      }
      return salida;
   }
   private String guardarJSon(List<Puntuacion> puntuaciones) {
      String string = "";
      try {
         JSONArray jsonArray = new JSONArray();
         for (Puntuacion puntuacion : puntuaciones) {
            JSONObject objeto = new JSONObject();
            objeto.put("puntos", puntuacion.getPuntos());
            objeto.put("nombre", puntuacion.getNombre());
            objeto.put("fecha", puntuacion.getFecha());
            jsonArray.put(objeto);
         }
         string = jsonArray.toString();
      } catch (JSONException e) {
          e.printStackTrace();
      }
      return string;
   }
   private List<Puntuacion> leerJSon(String string) {
      List<Puntuacion> puntuaciones = new ArrayList<>();
      try {
         JSONArray json_array = new JSONArray(string);
         for (int i = 0; i < json_array.length(); i++) {
```

```
            JSONObject objeto = json_array.getJSONObject(i);
            puntuaciones.add(new Puntuacion(objeto.getInt("puntos"),
                objeto.getString("nombre"), objeto.getLong("fecha")));
         }
      } catch (JSONException e) {
         e.printStackTrace();
      }
      return puntuaciones;
   }
}
```

2. Modifica el código necesario para que se pueda seleccionar este tipo de almacenamiento.
3. Si has realizado la práctica anterior, introduce los métodos `guardarString()` y `leerString()`.
4. Ejecuta el proyecto y verifica su funcionamiento.

NOTA: Acabamos de ver dos alternativas para serializar los datos contenidos en un objeto, XML y JSON. Existe otra alternativa aportada por en lenguaje Java, que consiste en implementar la interfaz `Serializable`[41]. *Aunque resulta muy sencillo de utilizar, también presenta algunos inconvenientes: La serialización ocupa mucho espacio (demasiado para transacciones por Internet), el formato obtenido es binario (no es visible o editable por un usuario) y solo se implementa en Java (no podemos interoperar con servidores con otros lenguajes).*



Preguntas de repaso: *Trabajando con JSON*

9.7. Bases de datos con SQLite

Las bases de datos son una herramienta de gran potencia en la creación de aplicaciones informáticas. Hasta hace muy poco resultaba costoso y complejo utilizar bases de datos en nuestras aplicaciones. No obstante, Android incorpora la librería SQLite, que nos permitirá utilizar bases de datos mediante el lenguaje SQL, de una forma sencilla y utilizando muy pocos recursos del sistema. Como verás en este apartado, almacenar tu información en una base de datos no es mucho más complejo que almacenarla en un fichero, y además resulta mucho más potente.



SQL es el lenguaje de programación más utilizado para bases de datos. No resulta complejo entender los ejemplos que se mostrarán en este libro. No obstante, si deseas hacer cosas más complicadas te recomiendo que consultes alguno de los muchos manuales que se han escrito sobre el tema.

---

[41] http://chuwiki.chuidiang.org/index.php?title=Serializaci%C3%B3n_de_objetos_en_java

Para manipular una base de datos en Android usaremos la clase abstracta `SQLiteOpenHelper`, que nos facilita tanto la creación automática de la base de datos como el trabajar con futuras versiones de esta base de datos. Para crear un descendiente de esta clase hay que implementar los métodos `onCreate()` y `onUpgrade()`, y opcionalmente, `onOpen()`. La gran ventaja de utilizar esta clase es que ella se preocupará de abrir la base de datos si existe, o de crearla si no existe. Incluso de actualizar la versión si decidimos crear una nueva estructura de la base de datos. Además, esta clase tiene dos métodos: `getReadableDatabase()` y `getWritableDatabase()`, que abren la base de datos en modo solo lectura o lectura y escritura. En caso de que todavía no exista la base de datos, estos métodos se encargarán de crearla.



Vídeo[tutorial]: *Uso de bases de datos en Android*



Ejercicio: *Utilizando una base de datos para guardar puntuaciones*

Pasemos a demostrar cómo guardar las puntuaciones obtenidas en Asteroides en una base de datos. Si comparas la solución propuesta con las anteriores, verás que el código necesario es menor. Además, una base de datos te da mucha más potencia; puedes, por ejemplo, ordenar la salida por puntuación, eliminar filas antiguas, etc. Todo esto sin aumentar apenas el uso de recursos.

1. Crea la clase `AlmacenPuntuacionesSQLite` en el proyecto Asteroides y escribe el siguiente código:

```
public class AlmacenPuntuacionesSQLite extends SQLiteOpenHelper
                                    implements AlmacenPuntuaciones{
   public AlmacenPuntuacionesSQLite(Context context) {
      super(context, "puntuaciones", null, 1);
   }

   //Métodos de SQLiteOpenHelper
   @Override public void onCreate(SQLiteDatabase db) {
      db.execSQL("CREATE TABLE puntuaciones ("+
            "_id INTEGER PRIMARY KEY AUTOINCREMENT, "+
            "puntos INTEGER, nombre TEXT, fecha BIGINT)");
   }

   @Override    public void onUpgrade(SQLiteDatabase db,
                              int oldVersion, int newVersion) {
   // En caso de una nueva versión habría que actualizar las tablas
   }

   //Métodos de AlmacenPuntuaciones
   public void guardarPuntuacion(int puntos, String nombre,
                                                long fecha) {
      SQLiteDatabase db = getWritableDatabase();
```

```
        db.execSQL("INSERT INTO puntuaciones VALUES ( null, "+
              puntos+", '"+nombre+"', "+fecha+")");
        db.close();
    }

    public List<String> listaPuntuaciones(int cantidad) {
        List<String> result = new ArrayList<String>();
        SQLiteDatabase db = getReadableDatabase();
        Cursor cursor = db.rawQuery("SELECT puntos, nombre FROM " +
         "puntuaciones ORDER BY puntos DESC LIMIT " +cantidad, null);
        while (cursor.moveToNext()){
              result.add(cursor.getInt(0)+" " +cursor.getString(1));
        }
        cursor.close();
        db.close();
        return result;
    }
}
```

El constructor de la clase se limita a llamar al constructor heredado con el perfil:

```
SQLiteOpenHelper(Context contexto, String nombre,
        SQLiteDatabase.CursorFactory cursor, int version).
```

Los parámetros se describen a continuación:

`contexto`: Contexto usado para abrir o crear la base de datos.

`nombre`: Nombre de la base de datos que se creará. En nuestro caso, "puntuaciones".

`cursor`: Se utiliza para crear un objeto de tipo cursor. En nuestro caso no lo necesitamos.

`version`: Número de versión de la base de datos empezando desde 1. En el caso de que la base de datos actual tenga una versión más antigua se llamará a `onUpgrade()`para que actualice la base de datos.

El método `onCreate()` se invocará cuando sea necesario crear la base de datos. Como parámetro se nos pasa una instancia de la base de datos que se acaba de crear. Este es el momento de crear las tablas que contendrán información. En nuestra aplicación necesitamos solo la tabla `puntuaciones`, que se crea por medio del comando SQL `CREATE TABLE puntuaciones...` El primer campo tiene por nombre `_id` y será un entero usado como clave principal. Su valor será introducido de forma automática por el sistema, de forma que dos registros no tengan nunca el mismo valor.

El método `onUpgrade()` está vacío. Si más adelante decidimos crear una nueva estructura para la base de datos, tendremos que indicar un número de versión superior, por ejemplo la 2. Cuando se ejecute el código sobre un sistema donde se disponga de una base de datos con la versión 1, se invocará el método `onUpgrade()`. En él tendremos que escribir los comandos necesarios para transformar la antigua base de datos en la nueva, tratando de conservar la información de la versión anterior.

Pasemos a describir los dos métodos de la interfaz `AlmacenaPuntuaciones`. El método `guardarPuntuacion()` comienza obteniendo una referencia a nuestra base de datos utilizando `getWritableDatabase()`, mediante la cual ejecuta el comando SQL para almacenar un nueva fila en la tabla `INSERT INTO puntuaciones...` El método `listaPuntuaciones()` comienza obteniendo una referencia a nuestra base de datos utilizando `getReadableDatabase()`. Realiza una consulta utilizando el método `rawQuery()`, con la que obtiene un cursor que utiliza para leer todas las filas devueltas en la consulta.

2. Modifica el código correspondiente para que este método pueda ser seleccionado para almacenar las puntuaciones.
3. Verifica su funcionamiento.



Ejercicio: *Verificación de los ficheros creados*

1. Selecciona la pestaña *Device File Explorer* (esquina inferior derecha).
2. Busca la ruta *data/data/org.example.asteroides.*
3. Observa los ficheros creados y compara el tamaño de cada uno.

Name	Size	Date	Time	Permissi
org.example.asteroides		2011-07-27	22:24	drwxr-xr-
databases		2011-09-23	10:32	drwxrwx-
puntuaciones	5120	2011-10-15	07:51	-rw-rw--
files		2011-09-23	10:03	drwxrwx-
puntuaciones.txt	28	2011-09-23	10:05	-rw-rw--
lib		2011-07-27	22:24	drwxr-xr-
shared_prefs		2011-09-21	12:14	drwxrwx-
org.example.asteroides_preferences.xml	215	2011-09-22	20:49	-rw-rw--

9.7.1. Los métodos *query()* y *rawQuery()*

En el ejemplo anterior hemos utilizado el método `rawQuery()` para hacer una consulta. Este método tiene una versión alternativa con la misma función: el método `query()`. El método `query()` es el usado por defecto en la documentación oficial y además es el único disponible en otras clases (por ejemplo, para hacer una consulta en un ContentProvider). Sin embargo, tiene un inconveniente respecto al método `rawQuery()`: has de rellenar una gran cantidad de parámetros para controlar la búsqueda, lo que lo hace confuso de utilizar. Si estás acostumbrado a trabajar con SQL, es posible que este método te resulte incómodo. A continuación se describen los parámetros de ambos métodos:

```
Cursor SQLiteDatabase.query (
        String table,            //tabla a consultar (FROM)
        String[] columns,        //columnas a devolver (SELECT)
        String selection,        //consulta (WHERE)
        String[] selectionArgs,//reemplaza "?" de la consulta
        String groupBy,          //agrupado por (GROUPBY)
        String having,           //condición para agrupación
        String orderBy,          //ordenado por
```

```
        String limit)           //cantidad máx. de registros

Cursor SQLiteDatabase.rawQuery(
        String sql,             //comando SQL
        String[] selectionArgs)//reemplaza "?" de la consulta
```

Veamos un ejemplo de cómo se podrían utilizar estos métodos. Supongamos que hemos creado la tabla `tabla` y que tiene las columnas `texto`, `entero` y `numero`. Si quisiéramos seleccionar las columnas `texto` y `entero` de las filas con el valor de `numero` mayor que 2, ordenados según el valor de `entero` y que además el número de filas seleccionadas estuviera limitado a un máximo de `cantidad` (donde *cantidad* ha de ser una variable de tipo entero previamente definida), escribiríamos:

```
Cursor cursor = db.rawQuery("SELECT texto, entero FROM tabla" +
     " WHERE numero>2 ORDER BY entero LIMIT " + cantidad, null);
```

Cuando uno está acostumbrado al lenguaje SQL, esta puede ser la forma más sencilla de hacer la consulta. De forma alternativa podemos hacer uso del segundo parámetro. Este ha de ser un *array* de String, de forma que estos *strings* reemplacen cada una de las apariciones del carácter "?" en la cadena del primer parámetro. Veamos un ejemplo que sería equivalente al anterior:

```
String[] param = new String[1];
param[0]= Integer.toString(cantidad,10);
Cursor cursor = db.rawQuery("SELECT texto, entero FROM tabla" +
              " WHERE numero>2 ORDER BY entero LIMIT ?", param);
```

Si en lugar del método `rawQuery()` queremos utilizar el método `query()`, usaríamos el siguiente código equivalente a los dos anteriores:

```
String[] CAMPOS = {"texto", "entero"};
Cursor cursor = db.query("tabla", CAMPOS, "numero>2", null,
         null, null, "entero", Integer.toString(cantidad));
```

Ejercicio: *Utilización del método* query() *para guardar puntuaciones*

1. Reemplaza la llamada al método `rawQuery()` del ejercicio anterior por el siguiente código:

```
String[] CAMPOS = {"puntos", "nombre"};
Cursor cursor=db.query("puntuaciones", CAMPOS, null, null,
        null, null, "puntos DESC", Integer.toString(cantidad));
```

2. Verifica que el funcionamiento es idéntico.



Preguntas de repaso: *SQLite I*

9.7.2. Uso de bases de datos en Mis Lugares

En los próximos ejercicios pasamos a demostrar cómo guardar los datos de la aplicación Mis Lugares en una base de datos. Esta estará formada por una única tabla (`lugares`). A continuación, se muestran las columnas que contendrán y las filas que se introducirán como ejemplo. Los valores que aparecen en las columnas `_id` y `fecha` no coincidirán con los valores reales:

_id	nombre	direccion	longitud	latitud	tipo	foto	telefono	url	Comentario	fecha	valoracion
1	Escuela	C/ Paran	-0.166	38.99	7		962849	ht	Uno de lo	2345	3.0
2	Al de	P. Indust	-0.190	38.92	2		636472	ht	No te pier	2345	3.0
4	android	ciberesp	0.0	0.0	7			ht	Amplia tu	2345	5.0
7	Barranc	Vía Verd	-0.295	38.86	9			ht	Espectacu	2345	4.0
5	La Vital	Avda. de	-0.172	38.97	6		962881	ht	El típico c	2345	2.0

Tabla 10: *Estructura de la tabla `Lugares` de la base de datos `Lugares`.*



Ejercicio: *Utilizando una base de datos en Mis Lugares*

1. Comenzamos haciendo una copia del proyecto, dado que en la nueva versión se eliminará parte del código desarrollado y es posible que quieras consultarlo en un futuro. Abre en el explorador de ficheros la carpeta que contiene el proyecto. Para hacer esto, puedes pulsar con el botón derecho sobre *app* en el explorador del proyecto y seleccionar *Show in Explorer*. Haz una copia de esta carpeta con un nuevo nombre.

2. Crea la clase `LugaresBD`, en el paquete `datos`, con el siguiente código:

```
public class LugaresBD extends SQLiteOpenHelper {

   Context contexto;

   public LugaresBD(Context contexto) {
      super(contexto, "lugares", null, 1);
      this.contexto = contexto;
   }

   @Override public void onCreate(SQLiteDatabase bd) {
      bd.execSQL("CREATE TABLE lugares ("+
              "_id INTEGER PRIMARY KEY AUTOINCREMENT, "+
              "nombre TEXT, " +
              "direccion TEXT, " +
              "longitud REAL, " +
              "latitud REAL, " +
              "tipo INTEGER, " +
              "foto TEXT, " +
              "telefono INTEGER, " +
              "url TEXT, " +
              "comentario TEXT, " +
              "fecha BIGINT, " +
              "valoracion REAL)");
```

```
      bd.execSQL("INSERT INTO lugares VALUES (null, "+
        "'Escuela Politécnica Superior de Gandía', "+
        "'C/ Paranimf, 1 46730 Gandia (SPAIN)', -0.166093, 38.995656, "+
        TipoLugar.EDUCACION.ordinal() + ", '', 962849300, "+
        "'http://www.epsg.upv.es', "+
        "'Uno de los mejores lugares para formarse.', "+
        System.currentTimeMillis() +", 3.0)");
      bd.execSQL("INSERT INTO lugares VALUES (null, 'Al de siempre', "+
        "'P.Industrial Junto Molí Nou - 46722, Benifla (Valencia)', "+
        " -0.190642, 38.925857, " + TipoLugar.BAR.ordinal() + ", '', "+
        "636472405, '', "+"'No te pierdas el arroz en calabaza.', " +
        System.currentTimeMillis() +", 3.0)");
      bd.execSQL("INSERT INTO lugares VALUES (null, 'androidcurso.com', "+
        "'ciberespacio', 0.0, 0.0,"+TipoLugar.EDUCACION.ordinal()+", '', "+
        "962849300, 'http://androidcurso.com', "+
        "'Amplia tus conocimientos sobre Android.', "+
        System.currentTimeMillis() +", 5.0)");
    bd.execSQL("INSERT INTO lugares VALUES (null,'Barranco del Infierno',"+
        "'Vía Verde del río Serpis. Villalonga (Valencia)', -0.295058, "+
        "38.867180, "+TipoLugar.NATURALEZA.ordinal() + ", '', 0, "+
       "'http://sosegaos.blogspot.com.es/2009/02/lorcha-villalonga-via-verde-del-"+
       "rio.html', 'Espectacular ruta para bici o andar', "+
       System.currentTimeMillis() +", 4.0)");
      bd.execSQL("INSERT INTO lugares VALUES (null, 'La Vital', "+
       "'Avda. La Vital,0 46701 Gandia (Valencia)',-0.1720092,38.9705949,"+
        TipoLugar.COMPRAS.ordinal() + ", '', 962881070, "+
        "'http://www.lavital.es', 'El típico centro comercial', "+
        System.currentTimeMillis() +", 2.0)");
   }

   @Override public void onUpgrade(SQLiteDatabase db, int oldVersion,
                                                     int newVersion) {
   }
}
```

```
class LugaresBD(val contexto: Context) :
                          SQLiteOpenHelper(contexto, "lugares", null, 1) {
   override fun onCreate(bd: SQLiteDatabase) {
     bd.execSQL("CREATE TABLE lugares (" +
          "_id INTEGER PRIMARY KEY AUTOINCREMENT, " +
          "nombre TEXT, " +
          "direccion TEXT, " +
          "longitud REAL, " +
          "latitud REAL, " +
          "tipo INTEGER, " +
          "foto TEXT, " +
          "telefono INTEGER, " +
          "url TEXT, " +
          "comentario TEXT, " +
          "fecha BIGINT, " +
          "valoracion REAL)")
    bd.execSQL(("INSERT INTO lugares VALUES (null, " +
       "'Escuela Politécnica Superior de Gandía', " +
       "'C/ Paranimf, 1 46730 Gandia (SPAIN)', -0.166093, 38.995656, "+
```

```
        TipoLugar.EDUCACION.ordinal + ", '', 962849300, " +
        "'http://www.epsg.upv.es', " +
        "'Uno de los mejores lugares para formarse.', " +
        System.currentTimeMillis() + ", 3.0)"))
    bd.execSQL(("INSERT INTO lugares VALUES (null, 'Al de siempre', " +
        "'P.Industrial Junto Molí Nou - 46722, Benifla (Valencia)', " +
        " -0.190642, 38.925857, " + TipoLugar.BAR.ordinal + ", '', " +
        "636472405, '', " + "'No te pierdas el arroz en calabaza.', " +
        System.currentTimeMillis() + ", 3.0)"))
    bd.execSQL(("INSERT INTO lugares VALUES (null, 'androidcurso.com', "+
       "'ciberespacio', 0.0, 0.0,"+TipoLugar.EDUCACION.ordinal+", '', "+
       "962849300, 'http://androidcurso.com', " +
       "'Amplia tus conocimientos sobre Android.', " +
       System.currentTimeMillis() + ", 5.0)"))
    bd.execSQL(("INSERT INTO lugares VALUES (null,'Barranco del Infierno',"+
       "'Vía Verde del río Serpis. Villalonga (Valencia)', -0.295058, "+
       "38.867180, " + TipoLugar.NATURALEZA.ordinal + ", '', 0, " +
       "'http://sosegaos.blogspot.com.es/2009/02/lorcha-villalonga-via-verde-del-"+
       "rio.html', 'Espectacular ruta para bici o andar', " +
       System.currentTimeMillis() + ", 4.0)"))
    bd.execSQL(("INSERT INTO lugares VALUES (null, 'La Vital', " +
       "'Avda. La Vital,0 46701 Gandia (Valencia)',-0.1720092,38.9705949,"+
       TipoLugar.COMPRAS.ordinal + ", '', 962881070, " +
       "'http://www.lavital.es', 'El típico centro comercial', " +
       System.currentTimeMillis() + ", 2.0)"))
  }

  override fun onUpgrade(db: SQLiteDatabase, oldVersion: Int,
                                            newVersion: Int) {}
}
```

El constructor de la clase se limita a llamar al constructor heredado. Los parámetros se describen a continuación:

`contexto`: Contexto usado para abrir o crear la base de datos.

`nombre`: Nombre de la base de datos que se creará. En nuestro caso, "puntuaciones".

`cursor`: Se utiliza para crear un objeto de tipo cursor. En nuestro caso no lo necesitamos.

`version`: Número de versión de la base de datos empezando desde 1. En el caso de que la base de datos actual tenga una versión más antigua se llamará a `onUpgrade()`para que actualice la base de datos.

El método `onCreate()` se invocará cuando sea necesario crear la base de datos. Como parámetro se nos pasa una instancia de la base de datos que se acaba de crear. Este es el momento de crear las tablas que contendrán información. El primer campo tiene por nombre `_id` y será un entero usado como clave principal. Su valor será introducido de forma automática por el sistema, de forma que dos registros no tengan nunca el mismo valor.

En nuestra aplicación necesitamos solo la tabla `lugares`, que es creada por medio del comando SQL `CREATE TABLE lugares...` La primera columna tiene por

nombre _id y será un entero usado como clave principal. Su valor será introducido automáticamente por el sistema, de forma que dos filas no tengan nunca el mismo valor de _id.

Las siguientes líneas introducen nuevas filas en la tabla utilizando el comando SQL `INSERT INTO lugares VALUES( , , ... )`. Los valores deben introducirse en el mismo orden que las columnas. La primera columna se deja como `null` dado que corresponde al `_id` y es el sistema quien ha de averiguar el valor correspondiente. Los valores de tipo `TEXT` deben introducirse entre comillas, pudiendo utilizar comillas dobles o simples. Como en Java se utilizan comillas dobles, en SQL utilizaremos comillas sencillas. El valor `TipoLugar.EDUCACION.ordinal()` corresponde a un entero según el orden en la definición de este enumerado y `System.currentTimeMillis()` corresponde a la fecha actual representada como número de milisegundos transcurridos desde 1970. El resto de los valores son sencillos de interpretar.

Ha de quedar claro que este constructor solo creará una base de datos (llamando a `onCreate()`) si todavía no existe. Si ya fue creada en una ejecución anterior, nos devolverá la base de datos existente.

El método `onUpgrade()` está vacío. Si más adelante, en una segunda versión de Mis Lugares, decidiéramos crear una nueva estructura para la base de datos, tendríamos que indicar un número de versión superior, por ejemplo, la 2. Cuando se ejecute el código sobre un sistema que disponga de una base de datos con la versión 1, se invocará el método `onUpgrade()`. En él tendremos que escribir los comandos necesarios para transformar la antigua base de datos en la nueva, tratando de conservar la información de la versión anterior.

3. Para acceder a los datos de la aplicación se definió la interfaz `RepositorioLugares`. Vamos a implementar esta interfaz para que los cambios sean los mínimos posibles. Añade el texto subrayado a la clase:

```
public class LugaresBD extends SQLiteOpenHelper
                        implements RepositorioLugares {
```

```
class LugaresBD(val contexto: Context) :
    SQLiteOpenHelper(contexto, "lugares", null, 1), RepositorioLugares {
```

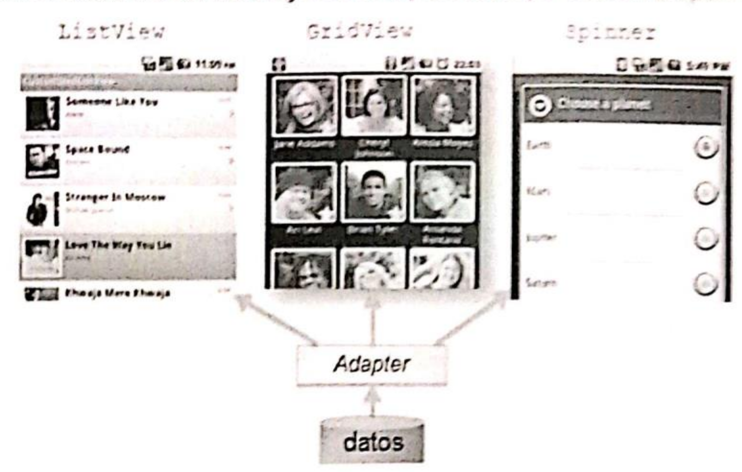
Aparecerá un error justo en la línea que acabas de introducir. Si sitúas el cursor de texto sobre el error, aparecerá una bombilla roja con opciones para resolver el error. Pulsa en *"Implement methods / members"*, selecciona todos los métodos y pulsa OK. Observa cómo en la clase se añaden todos los métodos de esta interfaz. De momento vamos a dejar estos métodos sin implementar. En la sección "Operaciones con bases de datos en Mis Lugares" aprenderemos a realizar las operaciones básicas cuando trabajemos con datos: altas, bajas, modificaciones y consultas.



4. No ejecutes todavía la aplicación. Hasta que no hagamos el siguiente ejercicio no funcionará correctamente.

9.7.3. Adaptadores para bases de datos

Un adaptador (`Adapter`) es un mecanismo de Android que hace de puente entre nuestros datos y las vistas contenidas en un `RecyclerView`, `ListView`, `GridView` o `Spinner`.



En el siguiente ejercicio vamos a crear un adaptador que toma la información de la base de datos que acabamos de crear y se la muestra a un `RecyclerView`. Realmente podríamos usar el adaptador `AdaptadorLugares` que ya tenemos creado. Este adaptador toma la información de un objeto que sigue la interfaz `RepositorioLugares`, restricción que cumple la clase `LugaresBD`. No obstante, vamos a realizar una implementación alternativa. La razón es que la implementación actual de `AdaptadorLugares` necesitaría una consulta a la base de datos cada vez que requiera una información de `Lugares`. Veremos más adelante que cada llamada a `elemento()`, `añade()`, `nuevo()`... va a suponer un acceso a la base de datos.

El nuevo adaptador, `AdaptadorLugaresBD`, va a trabajar de una forma más eficiente. Vamos a realizar una consulta de los elementos a listar y los va a guardar en un objeto de la clase `Cursor`. Mantendrá esta información mientras no cambie la información a listar, por lo que solo va a necesitar una consulta a la base de datos. En la aplicación, el `Cursor` será cargado al mostrar el listado inicial en `MainActivity`. Cuando el usuario quiera mostrar el detalle de algún lugar, no será necesario hacer una nueva consulta a la base de datos, dado que la información ya está en el `Cursor`.



Ejercicio: *Un Adaptador para base de datos en Mis Lugares*

1. Crea la clase `AdaptadorLugaresBD`, en el paquete `presentacion`, con el código:

```
public class AdaptadorLugaresBD extends AdaptadorLugares {

    protected Cursor cursor;

    public AdaptadorLugaresBD(RepositorioLugares lugares, Cursor cursor) {
        super(lugares);
        this.cursor = cursor;
    }
```

```
    public Cursor getCursor() {
        return cursor;
    }

    public void setCursor(Cursor cursor) {
        this.cursor = cursor;
    }

    public Lugar lugarPosicion(int posicion) {
        cursor.moveToPosition(posicion);
        return LugaresBD.extraeLugar(cursor);
    }

    public int idPosicion(int posicion) {
        cursor.moveToPosition(posicion);
        if (cursor.getCount()>0) return cursor.getInt(0);
        else                     return -1;
    }

    @Override
    public void onBindViewHolder(ViewHolder holder, int posicion) {
        Lugar lugar = lugarPosicion(posicion);
        holder.personaliza(lugar);
        holder.itemView.setTag(new Integer(posicion));
    }

    @Override public int getItemCount() {
        return cursor.getCount();
    }
}
```

```
class AdaptadorLugaresBD(lugares: LugaresBD, var cursor: Cursor)
                                          :AdaptadorLugares(lugares){
   fun lugarPosicion(posicion: Int): Lugar {
      cursor.moveToPosition(posicion)
      return (lugares as LugaresBD).extraeLugar(cursor)
   }
   fun idPosicion(posicion: Int): Int {
      cursor.moveToPosition(posicion)
      if (cursor.count>0) return cursor.getInt(0)
      else                return -1
   }

   override fun onBindViewHolder(holder: AdaptadorLugares.ViewHolder,
                                                   posicion: Int) {
      val lugar = lugarPosicion(posicion)
      holder.personaliza(lugar, onClick)
      holder.view.tag = posicion
   }

   override fun getItemCount(): Int {
      return cursor.getCount()
   }
}
```

Esta clase extiende AdaptadorLugares; de esta forma aprovechamos parte del código del adaptador y solo tenemos que indicar las diferencias. La más importante es que ahora el constructor tiene un nuevo parámetro de tipo Cursor, que es el resultado de una consulta en la base de datos. Realmente es aquí donde vamos a buscar los elementos a listar en el RecyclerView. Esta forma de trabajar es mucho más versátil que utilizar un *array*, podemos listar un tipo de lugar o que cumplan una determinada condición sin más que realizar un pequeño cambio en la consulta SQL. Además, podremos ordenarlos por valoración o cualquier otro criterio, y se mostrarán en el mismo orden como aparecen en el Cursor. Por otra parte, resulta muy eficiente dado que se realiza solo una consulta a la base de datos, dejando el resultado almacenado en la variable Cursor.

En Java el constructor de la clase se limita a llamar al super() y a almacenar el nuevo parámetro en una variable global. **En Kotlin** este proceso se indica en la declaración. **En Java** se han añadido los métodos *getter* y *setter* que permiten acceder al Cursor desde fuera de la clase.

Con el método lugarPosicion() vamos a poder acceder a un lugar, indicar su posición en Cursor o, lo que es lo mismo, su posición en el listado. Para ello, movemos el cursor a la posición indicada y extraemos el lugar en esta posición, utilizando un método estático de LugarBD.

Cuando queramos realizar una operación de borrado o edición de un registro de la base de datos, vamos a identificar el lugar a modificar por medio de la columna _id. Recuerda que esta columna ha sido definida en la posición 0. Para obtener el id de un lugar conociendo la posición que ocupa en el listado, se ha definido el método idPosicion().

Los dos últimos métodos ya existen en la clase que extendemos, pero los vamos a reemplazar; por esta razón tienen la etiqueta de override. El primero es onBindViewHolder() que se utilizaba para personalizar la vista ViewHolder en una determinada posición. La gran diferencia entre el nuevo método es que ahora el lugar lo obtenemos del cursor, mientras que en el método anterior se obtenía de lugares. Esto supondría una nueva consulta en la base de datos por cada llamada a onBindViewHolder(), lo que sería muy poco eficiente. En el método getItemCount() pasa algo similar, obtener el número de elementos directamente del cursor es más eficiente que hacer una nueva consulta.

Observa la última línea de onBindViewHolder() (holder.view.tag = posicion). El atributo Tag permite asociar a una vista cualquier objeto con información extra. La idea es asociar a cada vista del RecyclerView la posición que ocupa en el listado. Así, cuando asociamos un onClickListener este nos indica la vista pulsada, pero no la posición. De esta forma, sabiendo la vista conoceremos su posición. En la implementación anterior usábamos un método alternativo: posición = recyclerView.getChildAdapterPosition(vista). Pero tiene el inconveniente de necesitar el recyclerView. Y ahora no vamos a disponer de él.

En Kotlin tanto las clases como los atributos son por defecto cerrados[42]. Por lo tanto, aparece un error al intentar heredar de `AdaptadorLugares`. Para resolverlo pulsa sobre la bombilla roja y selecciona *Make AdaptadorLugares Open*.

2. Añade a la clase `LugaresBD` los siguientes métodos:

```
public static Lugar extraeLugar(Cursor cursor) {
     Lugar lugar = new Lugar();
     lugar.setNombre(cursor.getString(1));
     lugar.setDireccion(cursor.getString(2));
     lugar.setPosicion(new GeoPunto(cursor.getDouble(3),
             cursor.getDouble(4)));
     lugar.setTipo(TipoLugar.values()[cursor.getInt(5)]);
     lugar.setFoto(cursor.getString(6));
     lugar.setTelefono(cursor.getInt(7));
     lugar.setUrl(cursor.getString(8));
     lugar.setComentario(cursor.getString(9));
     lugar.setFecha(cursor.getLong(10));
     lugar.setValoracion(cursor.getFloat(11));
     return lugar;
}

public Cursor extraeCursor() {
     String consulta = "SELECT * FROM lugares";
     SQLiteDatabase bd = getReadableDatabase();
     return bd.rawQuery(consulta, null);
}
```

```
fun extraeLugar(cursor: Cursor) = Lugar(
    nombre = cursor.getString(1),
    direccion = cursor.getString(2),
    posicion = GeoPunto(cursor.getDouble(3), cursor.getDouble(4)),
    tipoLugar = TipoLugar.values()[cursor.getInt(5)],
    foto = cursor.getString(6),
    telefono = cursor.getInt(7),
    url = cursor.getString(8),
    comentarios = cursor.getString(9),
    fecha = cursor.getLong(10),
    valoracion = cursor.getFloat(11) )

fun extraeCursor(): Cursor =
    readableDatabase.rawQuery("SELECT * FROM lugares",null)
```

El primer método crea un nuevo lugar con los datos de la posición actual de un `Cursor`. El segundo nos devuelve un cursor que contiene todos los datos de la tabla.

3. Abre la clase `Aplicacion` y reemplaza la declaración de la variable `lugares` y `adaptador`:

```
public RepositorioLugaresBD lugares;
public AdaptadorLugaresBD adaptador;
```

[42] https://youtu.be/sMbrh3-1ufA



```
@Override public void onCreate() {
   super.onCreate();
   lugares = new LugaresBD(this);
   adaptador= new AdaptadorLugaresBD(lugares, lugares.extraeCursor());
}
```

```
val lugares = LugaresBD(this)
val adaptador by lazy {AdaptadorLugaresBD(lugares,lugares.extraeCursor())}
```

Hemos cambiado la declaración de `lugares` y `adaptador` para utilizar las nuevas clases. **En Kotlin** la inicialización de las propiedades se recomienda realizarla en su declaración. Observa como para `adaptador` se utiliza `by lazy`, para indicar que la inicialización se realice cuando vallamos a utilizar la variable. De hacerlo inmediatamente corremos el peligro de que la base de datos no esté creada.

4. **En Java**, modifica las siguientes propiedades de `MainActivity`:

```
private RepositorioLugaresBD lugares;
private AdaptadorLugaresBD adaptador;
```

5. Reemplaza en `MainActivity` dentro de `onCreate()` el código subrayado:

```
adaptador.setOnItemClickListener(new View.OnClickListener() {
   @Override public void onClick(View v) {
      int pos =(Integer)(v.getTag());
      usoLugar.mostrar(pos);
   }
});
```

```
adaptador.onClick = {
   val pos = it.tag as Int
   usoLugar.mostrar(pos)
}
```

6. Ejecuta la aplicación y verifica que la lista se muestra correctamente. Si pulsas sobre un lugar se producirá un error.



Ejercicio: *Una caché para evitar accesos a los datos*

Cuando la información que visualiza la aplicación se almacena en un servidor externo (puede ser en la nube o una base de datos) hay que tratar de minimizar el número de acceso al servidor. En estos casos, es frecuente almacenar la información en una memoria local, para evitar accesos en caso de volver a necesitarla. Esta técnica se conoce en informática como caché.

En este ejercicio vamos a crear una clase inspirada en este concepto. Realmente no crearemos una estructura de datos para implementar la caché, si no que aprovecharemos que tenemos los datos en el adaptador para no hacer nuevos accesos.

1. Crea la clase `LugaresDBAdapter` con el siguiente código:

```
public class LugaresBDAdapter extends LugaresBD {

   private AdaptadorLugaresBD adaptador;

   public LugaresBDAdapter(Context contexto) {
      super(contexto);
   }

   public Lugar elementoPos(int pos) {
      return adaptador.lugarPosicion (pos);
   }
}
```

```
class LugaresBDAdapter(val contexto: Context) : LugaresBD(contexto) {
   val adaptador: AdaptadorLugaresBD

   fun elementoPos(pos: Int) = adaptador.lugarPosicion(pos)
}
```

Al extender de `LugaresDB` conseguimos que herede el comportamiento de `RepositorioLugares`. Añadimos el atributo `adaptador` que es la estructura que actuará como cache. El constructor se limita a llamar al constructor del padre. De momento solo sobrescribimos un método más, `elementoPos()`, que devolverá un elemento dada su posición.

Si te fijas hemos repartido el código en dos clases. En `LugaresDB` resolveremos el acceso a la base de datos y en `LugaresDBAdapter` añadimos la caché utilizando un adaptador.

2. **En Java** añade el *getter* y el *setter* para `adaptador` (opción *Generate... > Getter and Setter*).

3. En la clase `Aplicacion` reemplaza la declaración de la variable `lugares` para que sea de la nueva clase y añade al final de `onCreate()`:

```
lugares = new LugaresBDAdapter(this);
adaptador= new AdaptadorLugaresBD(lugares, lugares.extraeCursor());
lugares.setAdaptador(adaptador);
```

```
val lugares = LugaresBDAdapter(this)
...
lugares.adaptador = adaptador
```

4. **En Java**, modifica las clases `MainActivity`, `VistaLugarActivity`, `EdicionLugarActivity` y `CasosUsoLugar` haz que `lugares` sea de tipo `LugaresDBAdapter`. **En Kotlin**, no es necesario, el tipo no se indica al venir de la declaración en `Aplicacion`.

5. En `VistaLugarActivity`, `EdicionLugarActivity` y `CasosUsoLugar` modifica la siguiente línea:

```
lugar = Lugares.elementoPos(pos);
```

```
lugar = lugares.elementoPos(pos)
```

Al entrar en la vista de un lugar podemos obtenerlo del adaptador a partir de su posición.

6. Ejecuta la aplicación y verifica que funciona. Puedes seleccionar un lugar e incluso editarlo, aunque si guardas una edición no se almacenarán los cambios. Lo arreglaremos más adelante.

Ejercicio: *Adaptando la actividad del Mapa a LugaresDBAdapter*

En este ejercicio adaptaremos la actividad `MapaActivity` para que use adecuadamente la nueva forma de acceder a los lugares. Es decir, los lugares a mostrar en el mapa los obtendremos directamente del adaptador, en lugar de hacer una nueva consulta a la base de datos.

1. En `MapaActivity` modifica la siguiente propiedad. Solo es necesario **en Java**.

```
private RepositorioLugaresBDAdapter lugares;
```

2. Modifica las tres llamadas a `lugares.elemento()` por `lugares.elementoPos()`. Así los lugares son extraídos del adaptador.

3. En `LugaresBDAdapter` añade la siguiente función:

```
@Override public int tamaño(){
   return adaptador.getItemCount();
}
```

```
override fun tamaño(): Int = adaptador.itemCount
```

Sobrescribimos la función para que cuando trabajemos con un `LugaresBDAdapter`, el número total de elementos corresponda con los que se estén listando en el `RecyclerView`.

4. Verifica el funcionamiento de la actividad `MapaActivity`.



Práctica: *Probando consultas en Mis Lugares*

1. En el método `extraeCursor()` de la clase `LugaresBD` reemplaza el comando `SELECT * FROM lugares` por `SELECT * FROM lugares WHERE valoracion>1.0 ORDER BY nombre LIMIT 4`. Ejecuta la aplicación y verifica la nueva lista.

2. Realiza otras consultas similares. Si tienes dudas, puedes consultar en Internet la sintaxis del comando SQL `SELECT`.

3. Si quieres practicar el uso del método `query()`, puedes tratar de realizar una consulta utilizando este método.



Práctica: *Añadir criterios de ordenación y máximo en Preferencias*

1. Modifica el método `extraeCursor()` para que el criterio de ordenación y el máximo de lugares a mostrar corresponda con los valores que el usuario ha indicado en las preferencias.

2. Si el usuario escoge el primer criterio de ordenación has de dejar la consulta original sin introducir la cláusula "`ORDER BY`".

3. Si escoge el orden por valoración este ha de ser descendiente, de más valorados a menos. Puedes usar la cláusula "`ORDER BY valoracion DESC`".

4. Para ordenar por distancia puedes usar la siguiente consulta SQL:

```
"SELECT * FROM lugares ORDER BY " +
        "(" + lon + "-longitud)*(" + lon + "-longitud) + " +
        "(" + lat + "-latitud )*(" + lat + "-latitud )"
```

   Donde las variables `lon` y `lat` han de corresponder con la posición actual del dispositivo. Esta ecuación es una simplificación que no tiene en cuenta que los polos están achatados, pero funciona de forma adecuada.

5. Si no actualizamos el cursor con la lista el cambio de preferencias no será efectivo hasta que salgas de aplicación y vuelvas a entrar. Para evitar este incoveniente, llama a la actividad `PreferenciasActivity` mediante `startActivityForResult()`. En el método `onActivityResult()` has de actualizar el cursor de `adaptador` e indicar que todos los elementos han de redibujarse. Para esta última acción puedes utilizar `adaptador.notifyDataSetChanged()`.



Solución:

1. Reemplaza en `lugaresBD` el siguiente método:

```
public Cursor extraeCursor() {
    SharedPreferences pref =
            PreferenceManager.getDefaultSharedPreferences(contexto);
    String consulta;
    switch (pref.getString("orden", "0")) {
        case "0":
            consulta = "SELECT * FROM lugares ";
            break;
        case "1":
            consulta = "SELECT * FROM lugares ORDER BY valoracion DESC";
            break;
        default:
            double lon = ((Aplicacion) contexto.getApplicationContext())
                                        .posicionActual.getLongitud();
            double lat = ((Aplicacion) contexto.getApplicationContext())
                                        .posicionActual.getLatitud();
```

```
            consulta = "SELECT * FROM lugares ORDER BY " +
                    "(" + lon + "-longitud)*(" + lon + "-longitud) + " +
                    "(" + lat + "-latitud )*(" + lat + "-latitud )";
            break;
    }
    consulta += " LIMIT "+pref.getString("maximo","12");
    SQLiteDatabase bd = getReadableDatabase();
    return bd.rawQuery(consulta, null);
}
```

```
fun extraeCursor(): Cursor {
   val pref = PreferenceManager.getDefaultSharedPreferences(contexto)
   var consulta = when (pref.getString("orden", "0")) {
      "0" -> "SELECT * FROM lugares "
      "1" -> "SELECT * FROM lugares ORDER BY valoracion DESC"
      else -> {
         val lon = (contexto.getApplicationContext() as Aplicacion)
            .posicionActual.longitud
         val lat = (contexto.getApplicationContext() as Aplicacion)
            .posicionActual.latitud
         "SELECT * FROM lugares ORDER BY " +
                 "($lon - longitud)*($lon - longitud) + " +
                 "($lat - latitud )*($lat - latitud )"
      }
   }
   consulta += " LIMIT ${pref.getString("maximo", "12")}"
   return readableDatabase.rawQuery(consulta, null)
}
```

2. En `MainActivity` añade. *NOTA: Si utilizas casos de uso tendrás que adaptar este código.*

```
static final int RESULTADO_PREFERENCIAS = 0;

public void lanzarPreferencias(View view) {
   Intent i = new Intent(this, PreferenciasActivity.class);
   startActivityForResult(i, RESULTADO_PREFERENCIAS);
}

@Override protected void onActivityResult(int requestCode, int resultCode,
                                                            Intent data) {
   super.onActivityResult(requestCode, resultCode, data);
   if (requestCode == RESULTADO_PREFERENCIAS) {
      adaptador.setCursor(Lugares.extraeCursor());
      adaptador.notifyDataSetChanged();
   }
}
```

```
val RESULTADO_PREFERENCIAS = 0

fun lanzarPreferencias(view: View? = null) = startActivityForResult(
   Intent(this, PreferenciasActivity::class.java), RESULTADO_PREFERENCIAS)

override
```

```
fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
   super.onActivityResult(requestCode, resultCode, data)
   if (requestCode == RESULTADO_PREFERENCIAS) {
      adaptador.cursor = lugares.extraeCursor()
      adaptador.notifyDataSetChanged()
   }
}
```

Operaciones con bases de datos en Mis Lugares

En los apartados anteriores hemos aprendido a crear una base de datos y a realizar consultas en una tabla. En este apartado vamos a continuar aprendiendo las operaciones básicas cuando trabajamos con datos. Estas son: altas, bajas y modificaciones y consultas.



Ejercicio: *Consulta de un elemento en Mis Lugares*

1. Reemplaza en la clase `LugaresBD` en el método `elemento()` con el siguiente código. Su finalidad es buscar el lugar correspondiente a un `id` y devolverlo.

```
@Override public Lugar elemento(int id) {
   Cursor cursor = getReadableDatabase().rawQuery(
                          "SELECT * FROM lugares WHERE _id = "+id, null);
   try {
      if (cursor.moveToNext())
         return extraeLugar(cursor);
      else
         throw new SQLException("Error al acceder al elemento _id = "+id);
   } catch (Exception e) {
      throw e;
   } finally {
      if (cursor!=null) cursor.close();
   }
}
```

```
override fun elemento(id: Int): Lugar {
   val cursor = readableDatabase.rawQuery(
                          "SELECT * FROM lugares WHERE _id = $id", null)
   try {
      if (cursor.moveToNext())
         return extraeLugar(cursor)
      else
         throw SQLException("Error al acceder al elemento _id = $id")
   } catch (e:Exception) {
      throw e
   } finally {
      cursor?.close()
   }
}
```

Comenzamos obteniendo la referencia a la base de datos con la propiedad `readableDatabase`. Este objeto nos permitirá hacer consultas en la base de datos. Por medio del método `rawQuery()` se realiza una consulta en la tabla `lugares` usando el comando SQL `SELECT * FROM lugares WHERE _id =`... Este comando podría interpretarse como "selecciona todas las columnas de la tabla `lugares`, para la fila con el `id` indicado". El resultado de una consulta es un `Cursor` con la fila, si es encontrado, o en caso contrario, un `Cursor` vacío.

En la siguiente línea llamamos a `cursor.moveToNext()` para que el cursor pase a la siguiente fila encontrada. Como es la primera llamada estamos hablando del primer elemento. Devuelve `true` si lo encuentra y `false` si no. En caso de encontrarlo, llamamos a `extraeLugar()` para actualizar todos los atributos de `lugar` con los valores de la fila apuntada por el cursor. Si no lo encontramos lanzamos una excepción.

Es importante cerrar lo antes posibles el cursor por tener mucho consumo de memoria. Lo hacemos en la sección `finally` para asegurarnos que se realiza siempre, haya habido una excepción o no.



Ejercicio: *Modificación de un lugar*

Si tratas de modificar cualquiera de los lugares, observarás que los cambios no tienen efecto. Para que la base de datos sea actualizada, realiza el siguiente ejercicio:

1. Añade en la clase `LugaresBD`, en el siguiente método `actualiza()`. Su finalidad es reemplazar el lugar correspondiente al *id* indicado por un nuevo lugar.

```
@Override public void actualiza(int id, Lugar lugar) {
   getWritableDatabase().execSQL("UPDATE lugares SET" +
            "   nombre = '" + lugar.getNombre() +
            "', direccion = '" + lugar.getDireccion() +
            "', longitud = " + lugar.getPosicion().getLongitud() +
            " , latitud = " + lugar.getPosicion().getLatitud() +
            " , tipo = " + lugar.getTipo().ordinal() +
            " , foto = '" + lugar.getFoto() +
            "', telefono = " + lugar.getTelefono() +
            " , url = '" + lugar.getUrl() +
            "', comentario = '" + lugar.getComentario() +
            "', fecha = " + lugar.getFecha() +
            " , valoracion = " + lugar.getValoracion() +
            " WHERE _id = " + id);
}
```

```
override fun actualiza(id:Int, lugar:Lugar) = with(lugar) {
   writableDatabase.execSQL("UPDATE lugares SET " +
      "nombre = '$nombre', direccion = '$direccion', " +
      "longitud = ${posicion.longitud}, latitud = ${posicion.latitud}, " +
      "tipo = ${tipoLugar.ordinal}, foto ='$foto', telefono =$telefono, "+
      "url = '$url', comentario = '$comentarios', fecha = $fecha, " +
```

```
        "valoracion = $valoracion  WHERE _id = $id")
}
```

2. En la clase `EdicionLugarActivity`, en el método `onOptionsItemSelected()` añade el código subrayado y elimina el tachado:

```
int _id = lugares.getAdaptador().idPosicion(pos);
usoLugar.guardar(pos _id, lugar);
```

```
val _id = lugares.adaptador.idPosicion(pos)
usoLugar.guardar(pos _id, nuevoLugar)
```

La variable `pos` corresponde a un indicador de posición dentro de la lista. Para utilizar correctamente el método `actualiza()` de `LugaresBD`, hemos de obtener el `_id` correspondiente a la primera columna de la tabla. Este cambio lo realiza el método `idPosicion()` de `adaptador`.

3. Ejecuta la aplicación y trata de modificar algún lugar. Observa que, cuando realizas un cambio en un lugar, estos no parecen en `VistaLugarActivity` ni en `MainActivity`. Realmente sí que se han almacenado, el problema está en que el `adaptador` no se ha actualizado. Para verificar que los cambios sí que se han almacenado puedes cambiar el criterio de ordenación, y así actualizar el adaptador.

4. Para resolver el refresco en `MainActivity` has de sobrescribir el método `actuliza()` en `LugaresDBAdapter`.

```
@Override public void actualiza(int id, Lugar lugar) {
   super.actualiza(id,lugar);
   adaptador.setCursor(extraeCursor());
   adaptador.notifyDataSetChanged();
}
```

```
override fun actualiza(id:Int, lugar:Lugar) {
   super.actualiza(id,lugar)
   adaptador.setCursor(extraeCursor())
   adaptador.notifyDataSetChanged()
}
```

Empezamos llamando al `super` para que el lugar se actualice en la base de datos según se indica en `LugaresBD`. Luego creamos un nuevo cursor, dado que al cambiar el lugar es posible que cambie el orden de los elementos seleccionados. Finalmente, notificamos al adaptador que los datos han cambiado y que vuelva a pintar las vistas correspondientes del `RecyclerView`.

5. Ejecuta de nuevo la aplicación. Tras editar un lugar, los cambios se reflejan en `MainActivity` pero no en `VistaLugarActivity`.

6. Para resolver este problema has de añadir las siguientes líneas en `VistaLugarActivity`:

```
private int _id = -1;

@Override public void onCreate(…) {
   …
```

```
    pos = extras.getInt("pos", 0);
    _id = lugares.getAdaptador().idPosicion(pos);
    ...

@Override public void onActivityResult(...)
    if (requestCode == RESULTADO_EDITAR) {
       lugar = lugares.elemento(_id);
       pos = lugares.getAdaptador().posicionId(_id);
       actualizaVistas();
    }
```

```
private var _id: Int = -1

override fun onCreate(...) {
    ...
    pos = intent.extras?.getInt("pos", 0) ?: 0
    _id = lugares.adaptador.idPosicion(pos)
    ...
override fun onActivityResult(...)
    if (requestCode == RESULTADO_EDITAR) {
       lugar = lugares.elemento(_id)
       pos = lugares.adaptador.posicionId(_id)
       actualizaVistas()
    }...
```

Necesitamos actualizar la variable `lugar` dado que esta acaba de ser modificada. Extraerla según su posición en el listado es potencialmente peligroso, dado que esta posición puede cambiar dinámicamente. Por ejemplo, si ordenamos los lugares por orden alfabético y modificamos su inicial, posiblemente cambie su posición. Por el contrario, el `_id` de un lugar nunca puede cambiar. Hemos obtenido el `_id` al crear la actividad. Tras la edición del lugar, con este `_id`, obtenemos los nuevos valores para `lugar` y buscamos la nueva posición a partir de `_id`.

7. Para hacer la última acción añade en `AdaptadorLugaresBD` la siguiente función:

```
public int posicionId(int id) {
     int pos = 0;
     while (pos<getItemCount() && idPosicion(pos)!=id) pos++;
     if (pos >= getItemCount()) return -1;
     else                       return pos;
}
```

```
fun posicionId(id: Int): Int {
   var pos = 0
   while (pos < itemCount && idPosicion(pos) != id) pos++
   return if (pos >= itemCount) -1
          else                  pos
}
```

Como ves, se recorren todos los elementos del adaptador hasta encontrar uno con el mismo `id`. Si no es encontrado devolvemos -1.

8. Ejecuta la aplicación y verifica el nuevo funcionamiento.



Ejercicio: *Modificación valoración y fotografía de un lugar*

1. Algunos de los campos de un lugar no se modifican en la actividad `EdicionLugarActivity`; se modifican directamente en `VistaLugarActivity`. En concreto la valoración, la fotografía y más adelante añadiremos fecha y hora. Cuando se modifiquen estos campos, también habrá que almacenarlos de forma permanente en la base de datos. Empezaremos por la valoración. Añade en el método `actualizaVistas()` de `VistaLugarActivity` el código subrayado.

```
valoracion.setOnRatingBarChangeListener(null);
valoracion.setRating(lugar.getValoracion());
valoracion.setOnRatingBarChangeListener(
        new RatingBar.OnRatingBarChangeListener() {
           @Override
           public void onRatingChanged(RatingBar ratingBar,
                                 float valor, boolean fromUser) {
              lugar.setValoracion(valor);
              pos = lugares.actualizaPosLugar(pos, lugar);
           }
        });
```

```
valoracion.setOnRatingBarChangeListener { _, _, _ -> }
valoracion.setRating(lugar.valoracion)
valoracion.setOnRatingBarChangeListener { _, valor, _ ->
   lugar.valoracion = valor
   pos = lugares.actualizaPosLugar(pos, lugar)
}
```

Cuando el usuario cambie la valoración de un lugar se llamará a `onRatingChanged()`, donde actualizamos la valoración y llamamos a `actualizarLugares()`. Esta función llamará a `actualizaVistas()`, donde cambiamos `raingBar`, lo que provocará una llamada al escuchador, y así sucesivamente entrando en bucle. Para evitarlo antes de cambiar el valor desactivamos el escuchador. Si tenemos seleccionada la ordenación por valoración, al cambiarla puede cambiar la posición del lugar en la lista. Por si ha cambiado, volvemos a obtener la variable `pos`.

2. Añade la siguiente función a `LugaresBDAdapter`:

```
public int actualizaPosLugar(int pos, Lugar lugar) {
   int id = adaptador.idPosicion(pos);
   actualiza(id, lugar);
   return adaptador.posicionId(id); //devolvemos la nueva posición
}
```

```
fun actualizaPosLugar(pos: Int, lugar: Lugar): Int {
   val id = adaptador.idPosicion(pos)
   actualiza(id, lugar)
   return adaptador.posicionId(id)  //devolvemos la nueva posición
}
```

Primero obtenemos en la variable id el identificador del lugar. Para ello, vamos a usar la posición que el lugar ocupa en el listado. Con el id, ya podemos actualizar la base de datos. La función devuelve la nueva posición que el lugar tiene en el adaptador.

3. Para que los cambios en las fotografías se actualicen también has obtener el lugar de forma adecuada y llamar a actualizaPosLugar():

```
public void ponerFoto(int pos, String uri, ImageView imageView) {
   Lugar lugar = lugares.elementoPos(pos);
   lugar.setFoto(uri);
   visualizarFoto(lugar, imageView);
   lugares.actualizaPosLugar(pos, lugar);
}
```

```
fun ponerFoto(pos: Int, uri: String?, imageView: ImageView) {
   val lugar = lugares.elementoPos(pos)
   lugar.foto = uri ?: ""
   visualizarFoto(lugar, imageView)
   lugares.actualizaPosLugar(pos, lugar)
}
```

4. Verifica que tanto los cambios de valoración como de fotografía se almacenan correctamente.



Ejercicio: *Alta de un lugar*

En este ejercicio aprenderemos a añadir nuevos registros a la base de datos.

1. Reemplaza en la clase LugaresBD el método nuevo() por el siguiente. Su finalidad es crear un nuevo lugar en blanco y devolver el id del nuevo lugar.

```
@Override public int nuevo() {
   int _id = -1;
   Lugar lugar = new Lugar();
   getWritableDatabase().execSQL("INSERT INTO lugares (nombre, " +
       "direccion, longitud, latitud, tipo, foto, telefono, url, " +
       "comentario, fecha, valoracion) VALUES ('', '',  " +
       lugar.getPosicion().getLongitud() + ","+
       lugar.getPosicion().getLatitud() + ", "+ lugar.getTipo().ordinal()+
       ", '', 0, '', '', " + lugar.getFecha() + ", 0)");
    Cursor c = getReadableDatabase().rawQuery(
       "SELECT _id FROM lugares WHERE fecha = " + lugar.getFecha(), null);
    if (c.moveToNext()) _id = c.getInt(0);
    c.close();
    return _id;
}
```

```
override fun nuevo():Int {
   var _id = -1
   val lugar = Lugar(nombre = "")
```

```
    writableDatabase.execSQL("INSERT INTO lugares (nombre, direccion, " +
        "longitud, latitud, tipo, foto, telefono, url, comentario, " +
        "fecha, valoracion) VALUES ('', '', ${lugar.posicion.longitud}, " +
        "${lugar.posicion.latitud}, ${lugar.tipoLugar.ordinal}, '', 0, " +
        "'', '', ${lugar.fecha},0 )")
    val c = readableDatabase.rawQuery((
        "SELECT _id FROM lugares WHERE fecha = " + lugar.fecha), null)
    if (c.moveToNext()) _id = c.getInt(0)
    c.close()
    return _id
}
```

Comenzamos inicializando el valor del `_id` a devolver a -1. De esta manera, si hay algún problema este será el valor devuelto. Luego se crea un nuevo objeto `Lugar`. Si consultas el constructor de la clase, observarás que solo se inicializan `posicion`, `tipo` y `fecha`. El resto de los valores serán una cadena vacía para `String` y 0 para valores numéricos. Acto seguido, se crea una nueva fila con esta información. Los valores de texto y numéricos tampoco se indican, al inicializarse de la misma manera.

El método ha de devolver el `_id` del elemento añadido. Para conseguirlo se realiza una consulta buscando una fila con la misma fecha que acabamos de introducir.

2. Para la acción de añadir vamos a utilizar el botón flotante que tenemos desde la primera versión de la aplicación. Abre el fichero *res/layout/activity_main.xml* y reemplaza el icono aplicado a este botón:

```
<android.support.design.widget.FloatingActionButton
    …
    android:src="@android:drawable/ic_input_add"
    … />
```

3. Abre la clase `MainActivity` y dentro de `onCreate()` comenta el código tachado y añade el subrayado, para que se ejecute al pulsar el botón flotante:

```
fab.setOnClickListener(new View.OnClickListener() {
    @Override public void onClick(View view) {
        Snackbar.make(view,"Replace with your own action",Snackbar.LENGTH_LONG)
                .setAction("Action", null).show();
        usoLugar.nuevo();
    }
});
```

```
fab.setOnClickListener { view ->
    Snackbar.make(view, "Replace with your own action",Snackbar.LENGTH_LONG)
        .setAction("Action", null).show()
    usoLugar.nuevo()
}
```

4. Añade el siguiente caso de uso en `CasoUsoLugar`:

```
public void nuevo() {
    int id = lugares.nuevo();
    GeoPunto posicion = ((Aplicacion) actividad.getApplication())
```

```
            .posicionActual;
    if (!posicion.equals(GeoPunto.SIN_POSICION)) {
        Lugar lugar = lugares.elemento(id);
        lugar.setPosicion(posicion);
        lugares.actualiza(id, lugar);
    }
    Intent i = new Intent(actividad, EdicionLugarActivity.class);
    i.putExtra("_id", id);
    actividad.startActivity(i);
}
```

```
fun nuevo() {
    val _id = lugares.nuevo()
    val posicion = (actividad.application as Aplicacion).posicionActual
    if (posicion != GeoPunto.SIN_POSICION) {
        val lugar = lugares.elemento(_id)
        lugar.posicion = posicion
        lugares.actualiza(_id, lugar)
    }
    val i = Intent(actividad, EdicionLugarActivity::class.java)
    i.putExtra("_id", _id)
    actividad.startActivity(i)
}
```

Comenzamos creando un nuevo lugar en la base de datos cuyo identificador va a ser `_id`. La siguiente línea obtiene la posición actual. Si el dispositivo está localizado, obtenemos el lugar recién creado, cambiamos su posición y lo volvemos a guardar. A continuación, vamos a lanzar la actividad `EdicionLugarActivity` para que el usuario rellene los datos del lugar. Hasta ahora hemos utilizado el extra "`pos`" para indicar la posición en la lista del objeto a editar. Pero ahora esto no es posible, dado que este nuevo lugar no ha sido añadido a la lista. Para resolver el problema vamos a crear un nuevo extra, "`_id`", que usaremos para identificar el lugar a editar por medio de su campo `_id`.

5. En la clase `EdicionLugarActivity` añade el código subrayado:

```
private int _id;

@Override protected void onCreate(Bundle savedInstanceState) {
    …
    Bundle extras = getIntent().getExtras();
    pos = extras.getInt("pos", -1);
    _id = extras.getInt("_id", -1);
    if (_id!=-1) lugar = lugares.elemento(_id);
    else         lugar = lugares.elementoPos(pos);
    actualizaVistas();
}
```

```
var _id = -1

override fun onCreate(savedInstanceState: Bundle?) {
    …
    pos = intent.extras?.getInt("pos", -1) ?: -1
    _id = intent.extras?.getInt("_id", -1) ?: -1
```

```
    lugar = if ( id !== -1) lugares.elemento( id)
            else            lugares.elementoPos(pos)
    actualizaVistas()
}
```

Esta actividad va a poder ser llamada de dos formas alternativas: usando el extra "pos" para indicar que el lugar a modificar ha de extraerse de una posición del adaptador; o usando "_id" en este caso el lugar será extraído de la base de datos usando su identificador. Observa cómo se han definido dos variables globales, pos e _id. Aunque solo una se va a inicializar y la otra valdrá -1.

6. Cuando el usuario pulse la opción guardar se llamará al método onOptionsItemSelected(). Para almacenar la información tendremos que verificar cuál de las dos variables ha sido inicializada. Añade el código subrayado en este método:

```
case R.id.accion_guardar:
    …
    if ( id==-1) int _id = lugares.getAdaptador().idPosicion(pos);
    usoLugar.guardar(_id, lugar);
    finish();
    return true;
```

```
R.id.accion_guardar -> {
    …
    if ( id==-1) val _id = lugares.adaptador.idPosicion(pos)
    usoLugar.guardar(_id, nuevoLugar)
    finish()
    return true
}
```

El if es añadido dado que si nos han pasado el identificador _id ya no tiene sentido obtenerlo a partir de la posición.

7. Verifica que los cambios introducidos funcionan correctamente.



Ejercicio: *Baja de un lugar*

En este ejercicio aprenderemos a eliminar filas de la base de datos.

1. Reemplaza en la clase LugaresBD el método borrar() por el siguiente. Su finalidad es eliminar el lugar correspondiente al id indicado.

```
public void borrar(int id) {
    getWritableDatabase().execSQL("DELETE FROM lugares WHERE _id = " + id);
}
```

```
override fun borrar(id: Int) {
    writableDatabase.execSQL("DELETE FROM lugares WHERE _id = $id")
}
```

2. Añade en la clase VistaLugarActivity, dentro del método onOptionsItemSelected(), el código subrayado:

```
case R.id.accion_borrar:
    usoLugar.borrarPos(pos)
    return true;
```

```
R.id.accion_borrar -> {
   usoLugar.borrarPos(pos)
   return true
}
```

3. En `CasosUsoLugar`, añade la función:

```
public void borrarPos(int pos) {
   int id = lugares.getAdaptador().idPosicion(pos);
   borrar(id);
}
```

```
fun borrarPos(pos: Int) {
   val id = lugares.adaptador.idPosicion(pos)
   borrar(id)
}
```

Tiene la misma finalidad que la función `borrar()`, pero indicando la posición.

4. Dentro de `borrar()`, añade las dos líneas subrayadas para actualizar el cursor y notificar al adaptador que los datos han cambiado:

```
Lugares.borrar(id);
Lugares.getAdaptador().setCursor(Lugares.extraeCursor());
Lugares.getAdaptador().notifyDataSetChanged();
actividad.finish();
```

```
lugares.borrar(id)
lugares.adaptador.cursor = lugares.extraeCursor()
lugares.adaptador.notifyDataSetChanged()
actividad.finish()
```

5. Ejecuta la aplicación y trata de dar de baja algún lugar.



Práctica: *Opción CANCELAR en el alta de un lugar*

Si seleccionas la opción *nuevo* y en la actividad `EdicionLugarActivity` seleccionas la opción *CANCELAR,* puedes verificar que esta opción funciona mal. Los datos introducidos no se guardarán; sin embargo, se creará un nuevo lugar con todos sus datos en blanco. Para verificarlo has de salir de la aplicación para que se recargue el adaptador.

Para evitar este comportamiento, borra el elemento nuevo cuando se seleccione la opción *CANCELAR*. Pero este comportamiento ha de ser diferente cuando el usuario entró en la actividad `EdicionLugarActivity` para editar un lugar ya existente. Para diferenciar estas dos situaciones puedes utilizar los extras `_id` y `pos`.



Solución:

Añade en el método `onOptionsItemSelected()` en la opción `accion_cancelar`:

```
if (_id!=-1) usoLugares.borrar(_id)
```

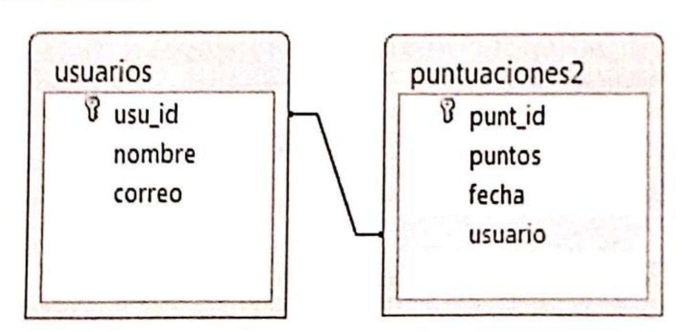


Preguntas de repaso: *SQLite II*

9.7.4. Bases de datos relacionales

Una base de datos relacional es aquella que está formada por varias tablas, de forma que se han establecido relaciones o conexiones entre alguno de sus campos. Esto permite que cuando nos situemos en una fila de una tabla se acceda de forma automática a la fila de otra tabla a través de esta relación.

Veamos un ejemplo: imaginemos que queremos almacenar varios datos sobre los usuarios de nuestro juego: nombre, correo electrónico, fecha del último acceso, etc. También queremos seguir guardando la puntuación obtenida en cada partida por cada usuario. La mejor solución sería almacenar esta información en dos tablas diferentes y crear una relación entre ellas. A continuación, se muestra un esquema de la estructura a definir:



Cada una de estas dos tablas comienza con un identificador numérico: *usu_id* y *punt_id*, que serán utilizados como código de usuario y código de puntuación, respectivamente. En la tabla `puntuaciones2`, además de la información propia de una puntuación se ha añadido el campo `usuario`. En este campo se ha de almacenar el código de usuario que obtuvo la puntuación. Se ha establecido una relación entre este campo y `usu_id` de la tabla `usuarios`. Esta forma de trabajar resulta muy eficiente: además de ahorrar memoria, evita que se repliquen los datos.

puntuaciones2

punt_id	puntos	fecha	usuario
1	10000	27/9/12	2
2	20000	28/9/12	2
3	10000	28/9/12	1

usuarios

usu_id	nombre	correo
1	Pepe	p@upv.es
2	Juan	j@upv.es



Ejercicio: *Una base de datos relacional para las puntuaciones*

Pasemos a demostrar cómo guardar las puntuaciones obtenidas en Asteroides en una base de datos relacional formada por dos tablas, tal y como se acaba de describir:

1. Crea la clase `AlmacenPuntuacionesSQLiteRel` en el proyecto `Asteroides`:

```
public class AlmacenPuntuacionesSQLiteRel extends SQLiteOpenHelper
                                          implements AlmacenPuntuaciones{
   public AlmacenPuntuacionesSQLiteRel(Context context) {
      super(context, "puntuaciones", null, 2);
   }

   //Métodos de SQLiteOpenHelper
   @Override public void onCreate(SQLiteDatabase db) {
      db.execSQL("CREATE TABLE usuarios ("+
            "usu_id INTEGER PRIMARY KEY AUTOINCREMENT, "+
            "nombre TEXT, correo TEXT)");
      db.execSQL("CREATE TABLE puntuaciones2 ("+
            "pun_id INTEGER PRIMARY KEY AUTOINCREMENT, "+
            "puntos INTEGER, fecha BIGINT, usuario INTEGER, "+
            "FOREIGN KEY (usuario) REFERENCES usuarios (usu_id))");
   }

   @Override public void onUpgrade(SQLiteDatabase db,
                              int oldVersion, int newVersion) {
      // En el siguiente ejercicio se implementará este método
   }
   //Métodos de AlmacenPuntuaciones
   public List<String> listaPuntuaciones(int cantidad) {
      List<String> result = new ArrayList<String>();
      SQLiteDatabase db = getReadableDatabase();
      Cursor cursor =db.rawQuery("SELECT puntos, nombre FROM "
         + "puntuaciones2, usuarios WHERE usuario = usu_id ORDER BY "
         + "puntos DESC LIMIT " + cantidad, null);
      while (cursor.moveToNext()){
            result.add(cursor.getInt(0)+" " +cursor.getString(1));
      }
      cursor.close();
      db.close();
```

```
      return result;
   }
   public void guardarPuntuacion(int puntos, String nombre,
               long fecha) {
      SQLiteDatabase db = getWritableDatabase();
      guardarPuntuacion(db, puntos, nombre, fecha);
      db.close();
   }
   public void guardarPuntuacion(SQLiteDatabase db, int puntos,
               String nombre, long fecha) {
      int usuario = buscaInserta(db, nombre);
      db.execSQL("PRAGMA foreign_keys = ON");
      db.execSQL("INSERT INTO puntuaciones2 VALUES ( null, " +
               puntos + ", " + fecha + ", " + usuario + ")");
   }
   private int buscaInserta(SQLiteDatabase db, String nombre) {
      Cursor cursor = db.rawQuery("SELECT usu_id FROM usuarios "
               + "WHERE nombre='" + nombre + "'", null);
      if (cursor.moveToNext()) {
         int result = cursor.getInt(0);
         cursor.close();
         return result;
      } else {
         cursor.close();
         db.execSQL("INSERT INTO usuarios VALUES (null, '" + nombre
               + "', 'correo@dominio.es')");
         return buscaInserta(db, nombre);
      }
   }
}
```

El constructor de la clase se limita a llamar al constructor heredado de forma similar al ejemplo anterior. La diferencia es que ahora indicamos que es la versión 2 en lugar de la 1.

El método `onCreate()` se invoca solo cuando no existe la base de datos. En nuestro caso se crean las dos tablas y se establece la relación mediante el código SQL `"FOREIGN KEY (usuario) REFERENCES usuarios (usu_id))"`.

El método `listaPuntuaciones()` es similar a la versión anterior, aunque se ha modificado ligeramente la consulta SQL. Ahora, la cláusula `FROM` incluye las dos tablas, además, se ha añadido `WHERE usuario = usu_id` para asegurarnos que se cumpla la relación. El método `guardarPuntuacion()` ha sido sobrecargado para disponer de dos versiones. En el siguiente ejercicio se clarificará por qué se ha actuado así. En la segunda versión se comienza llamando a `buscaInserta()` para obtener el código de usuario. Con esta información se crea una nueva entrada en la tabla `puntuaciones2`. Observa como antes se ha añadido el comando SQL `PRAGMA foreign_keys = ON`. Hay que ejecutarlo cada vez que abramos la base de datos para que el sistema realice una verificación automática de las relaciones definidas. Es decir, si se crea una nueva puntación

con un usuario que no existe en la tabla `usuarios` o se borra un usuario con puntuaciones; se producirá una excepción.

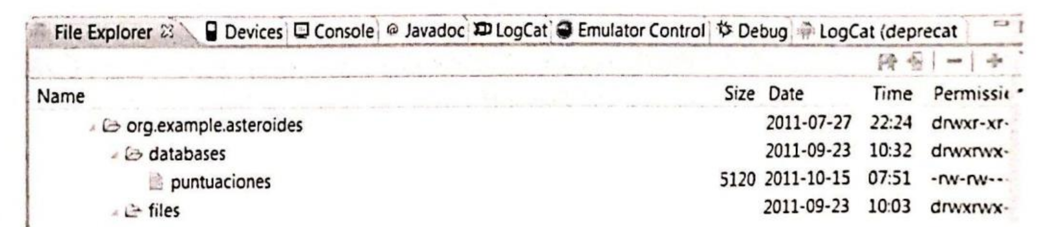
NOTA: Recuerda ejecutar este comando cada vez que abrás la base de datos.

Finalmente tenemos el método `buscaInserta()`. Comienza buscando el nombre de usuario. Si existe, devolverá su código; en caso contrario se ejecuta el comando SQL necesario para crear un nuevo usuario con este nombre. Tenemos que devolver el código del nuevo usuario. Esta información se obtiene realizando una llamada recursiva.

2. Abre el fichero *MainActivity.java* y modifica el método `onCreate()` para que se pueda ejecutar la siguiente línea:

```
almacen = new AlmacenPuntuacionesSQLiteRel(this);
```

3. Abre la vista *File Explorer* y busca la ruta */data/data/org.example.asteroides*.
4. Elimina los ficheros de la carpeta *databases*.



De no hacerlo no se llamaría al método `onCreate()`, al existir ya una base de datos. Esto ocasionaría un error, al no coincidir las tablas usadas en esta nueva versión con las de la base de datos. Una forma alternativa de eliminar las bases de datos es utilizar el administrador de aplicaciones del terminal, buscar la aplicación Asteroides y usar la opción *Borrar datos*. En el siguiente ejercicio veremos cómo evitar la necesidad de hacer esta tarea.

5. Verifica que las puntuaciones se almacenan correctamente.

9.7.5. El método *onUpgrade* de la clase SQLiteOpenHelper

El sistema Android facilita la actualización de las aplicaciones con una intervención mínima por parte del usuario. En este contexto, resulta muy importante que todos los datos introducidos por el usuario no se pierdan en una actualización.

Una nueva versión de nuestra aplicación suele requerir algún cambio en las estructuras de datos. Como hemos visto, la clase SQLiteOpenHelper dispone del método onUpgrade previsto con esta finalidad. El siguiente ejercicio nos muestra cómo utilizarlo.



Ejercicio: *Utilizando el método* onUpgrade *de la clase SQLiteOpenHelper*

1. Elimina el fichero con las bases de datos como se ha indicado en el punto 4 del ejercicio anterior.

2. Ejecuta Asteroides y selecciona la clase `AlmacenPuntuacionesSQLite` para el almacenamiento de datos. Sal de Asteroides.

3. Ejecuta Asteroides y destruye algún asteroide para obtener una puntuación. Verifica que se ha vuelto a crear el fichero de base de datos.

4. Abre la clase `AlmacenPuntuacionesSQLiteRel` y reemplaza el siguiente método:

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
   if (oldVersion==1 && newVersion==2){
      onCreate(db);                          //Crea las nuevas tablas
      Cursor cursor = db.rawQuery("SELECT puntos, nombre, fecha "+
               "FROM puntuaciones",null);     //Recorre la tabla antigua
      while (cursor.moveToNext()) {
         guardarPuntuacion(db, cursor.getInt(0), cursor.getString(1),
                                              cursor.getInt(2));
      }                                       //Crea los nuevos registros
      cursor.close();
      db.execSQL("DROP TABLE puntuaciones"); //Elimina tabla antigua
   }
}
```

Este código se ejecutará solo una vez, cuando el sistema detecta que se está instalando la versión 2 de la base de datos, pero se encuentra ya instalada la versión 1. Básicamente se crean las nuevas tablas (`onCreate(db)`), se recorre la tabla antigua para transportar la información a las nuevas y se termina borrando la tabla antigua. Recuerda que el número de versión se indica en el constructor.

5. Ejecuta Asteroides y selecciona la clase `AlmacenPuntuacionesSQLiteRel` para el almacenamiento de datos. Sal de Asteroides.

6. Ejecuta Asteroides (mejor si lo haces en modo *Debug* poniendo un *break point* al principio de este método) y muestra la lista de puntuaciones. Han de aparecer las mismas puntuaciones.

9.8. Content Provider

Los proveedores de contenido (ContentProviders) son uno de los bloques constructivos más importantes de Android. Nos van a permitir acceder a información proporcionada por otras aplicaciones, o a la inversa, compartir nuestra información con otras aplicaciones.

Tras describir los principios en los que se basan los ContentProviders, pasaremos a demostrar cómo acceder a ContentProviders creados por otras aplicaciones. Esta operación resulta muy útil, dado que te permitirá tener acceso a información interesante, como la lista de contactos, el registro de llamadas o los ficheros multimedia almacenados en el dispositivo. Terminaremos este apartado mostrando cómo crear tu propio ContentProvider, de forma que otras aplicaciones puedan acceder a tu información.

9.8.1. Conceptos básicos

El modelo de datos

La forma en la que un ContentProvider almacena la información es un aspecto de diseño interno, de forma que podríamos utilizar cualquiera de los métodos descritos en este capítulo. No obstante, cuando hagamos una consulta al ContentProvider se devolverá un objeto de tipo `Cursor`. Esto hace que la forma más práctica para almacenar la información sea en una base de datos.

Utilizando términos del modelo de bases de datos, un ContentProvider proporciona sus datos a través de una tabla simple, donde cada fila es un registro y cada columna es un tipo de datos con un significado particular. Por ejemplo, el ContentProvider que utilizaremos en el siguiente ejemplo se llama `CallLog`, y permite acceder al registro de llamadas del teléfono. La información que nos proporciona tiene la estructura que se muestra a continuación:

_id	Date	Number	Duration	Type
1	12/10/10 16:10	555123123	65	INCOMING_TYPE
3	12/11/10 20:42	555453455	356	OUTGOING_TYPE
4	13/11/10 12:15	555123123	90	MISSED_TYPE
5	14/11/10 22:10	555783678	542	OUTGOING_TYPE

Tabla 11: *Ejemplo de estructura de datos de un ContentProvider.*

Cada registro incluye el campo numérico `_id` que lo identifica de forma única. Como veremos a continuación, podremos utilizar este campo para identificar una llamada en concreto.

La forma de acceder a la información de un ContentProvider es muy similar al proceso descrito para las bases de datos. Es decir, vamos a poder realizar una consulta (incluso utilizando el lenguaje SQL), tras la cual se nos proporcionará un objeto de tipo `Cursor`. Este objeto contiene una información con una estructura similar a la mostrada en la tabla anterior.

Las URI

Para acceder a un ContentProvider en particular, será necesario identificarlo con una URI. Una URI (véase estándar RFC 2396) es una cadena de texto que permite identificar un recurso de información. Suele utilizarse frecuentemente en Internet (por ejemplo, para acceder a una página web). Una URI está formada por cuatro partes, tal y como se muestra a continuación:

```
<standard_prefix>://<authority>/<data_path>/<id>
```

La parte `<standard_prefix>` de todos los proveedores de contenido ha de ser siempre `content`. En el primer ejemplo de este apartado utilizaremos un ContentProvider donde Android almacena el registro de llamadas. Para acceder a este ContentProvider utilizaremos la siguiente URI:

```
content://call_log/calls
```

En la Tabla 12 encontrarás otros ejemplos de URI que te permitirán acceder a otras informaciones almacenadas en Android.

Para acceder a un elemento concreto has de indicar un `<id>` en la URI. Por ejemplo, si te interesa solo acceder a la llamada con identificador 4 (normalmente corresponderá a la cuarta llamada de la lista), has de indicar:

```
content://call_log/calls/4
```

Un mismo ContentProvider puede contener múltiples conjuntos de datos identificados por diferentes URI. Por ejemplo, para acceder a los ficheros multimedia almacenados en el móvil utilizaremos el ContentProvider `MediaStore`, utilizando algunos de los siguientes ejemplos de URI:

```
content://media/internal/images
content://media/external/video/5
content://media/*/audio
```

Cada ContentProvider suele definir constantes de `String` con sus correspondientes URI. Por ejemplo, `android.provider.CallLog.Calls.CONTENT_URI` corresponde a "`content://call_log/calls`". Y la constante `android.provider.MediaStore.Audio.Media.INTERNAL_CONTENT_URI` corresponde a "`content://media/internal/audio`".

9.8.2. Acceder a la información de un ContentProvider

Android utiliza los proveedores de contenido para almacenar diferentes tipos de información. Veamos los más importantes en la tabla siguiente:

Clase	Información almacenada	Ejemplos de URI
Browser	Enlaces favoritos, historial de navegación, historial de búsquedas.	content://browser/bookmarks
CallLog	Llamadas entrantes, salientes y perdidas.	content://call_log/calls
Contacts	Lista de contactos del usuario.	content://contacts/people
MediaStore	Ficheros de audio, vídeo e imágenes, almacenados en dispositivos de almacenamiento internos y externos.	content://media/internal/images content://media/external/video content://media/*/audio
Setting	Preferencias del sistema.	content://settings/system/ringtone content://settings/system/notificati on_sound
UserDictionary (a partir de 1.5)	Palabras definidas por el usuario, utilizadas en los métodos de entrada predictivos.	content://user_dictionary/words
Telephony (a partir de 1.5)	Mensajes SMS y MMS mandados o recibidos desde el teléfono.	content://sms content://sms/inbox content://sms/sent content://mms

Clase	Información almacenada	Ejemplos de URI
Calendar (a partir de 4.0)	Permite consultar y editar los eventos del calendario.	content://com.android.calendar/time content://com.android.calendar/events
Document (a partir de 4.4)	Permite acceder a ficheros locales o en la nube.	content://com.dropbox.android.Dropbox/metadata/ content://com.dominio.mio/dir/fich.txt

Tabla 12: *ContentProviders disponibles en Android.*

Leer información de un ContentProvider

Veamos un ejemplo que permite leer el registro de llamadas del teléfono. Crea una nueva aplicación y llámala ContentCallLog.

Reemplaza el contenido del fichero `res/layout/activity_main.xml` por:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView
      android:id="@+id/salida"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"/>
</LinearLayout>
```

De esta forma podremos identificar el `TextView` desde el programa y utilizarlo para mostrar la salida.

Añade en `AndroidManifest.xml` las líneas:

```
<uses-permission
      android:name="android.permission.READ_CALL_LOG"/>
```

Al solicitar el permiso `READ_CALL_LOG` podremos acceder al registro de llamadas. Añade al final del método `onCreate()` de la actividad principal el siguiente código:

```
...
String[] TIPO_LLAMADA = {"","entrante","saliente","perdida",
                  "mensaje de voz", "cancelada", "lista bloqueados"};
TextView salida = findViewById(R.id.salida);
Uri llamadas = Uri.parse("content://call_log/calls");
Cursor c = getContentResolver().query(llamadas,null,null,null,null);
while (c.moveToNext()) {
   salida.append("\n" + DateFormat.format("dd/MM/yy k:mm (",
               c.getLong(c.getColumnIndex(Calls.DATE)))
      + c.getString(c.getColumnIndex(Calls.DURATION)) + ") "
      + c.getString(c.getColumnIndex(Calls.NUMBER)) + ", "
      + TIPO_LLAMADA[Integer.parseInt(c.getString(
                              c.getColumnIndex(Calls.TYPE)))]);
}
```

En primer lugar se define un *array* de *strings*, de forma que `TIPO_LLAMADA[1]` corresponda a "`entrante`", `TIPO_LLAMADA[2]` corresponda a "`saliente`", etc. Luego se crea el objeto `salida`, que es asignado al `TextView` correspondiente del *layout*.

Ahora comienza lo interesante: creamos la URI `llamadas` asociada a `content://call_log/calls`. Para realizar la consulta creamos el `Cursor`, `c`. Se trata de la misma clase que hemos utilizado para hacer consultas en una base de datos. A través de `getContentResolver()`, obtenemos un `ContentResolver` asociado a la actividad, con el que podemos llamar al método `query()`. Este método permite varios parámetros para indicar exactamente los elementos que nos interesan, de forma similar a como se hace en una base de datos. Estos parámetros se estudiarán más adelante. Al no indicar ninguno se devolverán todas las llamadas registradas.

No tenemos más que desplazarnos por todos los elementos del cursor (`c.moveToNext()`) e ir añadiendo en la salida (`salida.append()`) la información correspondiente a cada registro. En concreto, fecha, duración, número de teléfono y tipo de llamada. Una vez que el cursor se encuentra situado en una fila determinada, podemos obtener la información de una columna utilizando los métodos `getString()`, `getInt()`, `getLong()` y `getFloat()`, dependiendo del tipo de dato almacenado. Estos métodos necesitan como parámetros el índice de columna. Para averiguar el índice de cada columna utilizaremos el método `getColumnIndex()`, indicando el nombre de la columna. En nuestro caso, estos nombres son "`date`", "`duration`", "`number`" y "`type`". En el ejemplo, en lugar de utilizar estos nombres directamente se han utilizado las cuatro constantes con estos valores en la clase `Calls`.

Especial atención merece la columna "`date`", que nos devuelve un entero largo que representa un instante concreto de una fecha. Para mostrarla en el formato deseado hemos utilizado el método estático `format()` de la clase `DateFormat`.

A continuación, se muestra el resultado de ejecutar este programa:

```
ContentCallLog
Hello World, ContentCallLog!
14/07/10 11:28 (455) 679314350, entrante
14/07/10 11:50 (84) 635478108, entrante
14/07/10 19:33 (0) 618614205, perdida
14/07/10 20:38 (0) 618614205, saliente
14/07/10 20:38 (30) 679314350, saliente
14/07/10 20:44 (35) 618614205, entrante
15/07/10 22:38 (371) 657638814, entrante
16/07/10 14:40 (66) 657638814, entrante
```

Veamos con más detalle el método `query()`:

```
Cursor query(Uri uri, String[] proyeccion, String seleccion,
             String[] argsSelecc, String orden)
```

Donde los parámetros corresponden a:

`uri`	URI correspondiente al ContentProvider a consultar.
`proyeccion`	Lista de columnas que queremos que nos devuelva.
`seleccion`	Cláusula SQL correspondiente a `WHERE`.

`argsSelecc`	Lista de argumentos utilizados en el parámetro `seleccion`.
`orden`	Cláusula SQL correspondiente a `ORDER BY`.

Para ilustrar el uso de estos parámetros, reemplaza en el ejemplo anterior:

```
Cursor c = getContentResolver.query(llamadas, null, null, null, null);
```

por:

```
String[] proyeccion = new String[] {
      Calls.DATE, Calls.DURATION, Calls.NUMBER, Calls.TYPE };
String[] argsSelecc = new String[] {"1"};
Cursor c = getContentResolver.query(
      llamadas,       // Uri del ContentProvider
      proyeccion,     // Columnas que nos interesan
      "type = ?",     // consulta WHERE
      argsSelecc,     // parámetros de la consulta anterior
      "date DESC");   // Ordenado por fecha, orden descendiente
```

De esta forma nos devolverán solo las columnas indicadas en `proyeccion`. Esto supone un ahorro de memoria en caso de que existan muchas columnas. En el siguiente parámetro se indican las filas que nos interesan por medio de una consulta de tipo `WHERE`. En caso de encontrar algún carácter `?`, este es sustituido por el primer `string` del parámetro `argsSelecc`. En caso de haber más caracteres `?` se irían sustituyendo siguiendo el mismo orden. Cuando se sustituyen los interrogantes, cada elemento de `argsSelecc` es introducido entre comillas. Por lo tanto, en el ejemplo la consulta `WHERE` resultante es "`WHERE type = '1'`". Esta consulta implica que solo se mostrarán las llamadas entrantes. El último parámetro sería equivalente a indicar en SQL "`SORTED BY date DESC`"; es decir, el resultado estará ordenado por fecha en orden descendiente.

Escribir información en un ContentProvider

Añadir un nuevo elemento en un ContentProvider resulta muy sencillo. Para ilustrar cómo se hace, escribe el siguiente código al principio del ejemplo anterior:

```
ContentValues valores = new ContentValues();
valores.put(Calls.DATE, new Date().getTime() );
valores.put(Calls.NUMBER, "555555555");
valores.put(Calls.DURATION, "55");
valores.put(Calls.TYPE, Calls.INCOMING_TYPE);
Uri nuevoElemento = getContentResolver().insert(
                                    Calls.CONTENT_URI, valores);
```

NOTA: Has de utilizar el `import java.util.Date`.

Como puedes ver, comenzamos creando un objeto `ContentValues`, donde vamos almacenado una serie de pares de valores, nombre de la columna y valor asociado a la columna. A continuación, se llama a `getContentResolver() .insert()` y se le pasa la URI del ContentProvider y los valores a insertar. Este método nos devuelve una URI que apunta de forma específica al elemento que acabamos de insertar. Podrías utilizar esta URI para hacer una consulta y obtener un cursor al nuevo elemento y así poder modificarlo, borrarlo u obtener el `_ID`. Recuerda que has

de pedir el permiso `WRITE_CALL_LOG`. Si ejecutas ahora el programa, la nueva llamada insertada ha de aparecer en primer lugar.



Estamos modificando el registro de llamadas del sistema; por lo tanto, también puedes verificar esta información desde las aplicaciones del sistema.



Borrar y modificar elementos de un ContentProvider

Puedes utilizar el método `delete()` para eliminar elementos de un ContentProvider:

```
int ContentProvider.delete(Uri uri, String seleccion, String[] argsSelecc)
```

Este método devuelve el número de elementos eliminados. Los tres parámetros del método se detallan a continuación:

`uri`	URI correspondiente al ContentProvider a consultar.
`seleccion`	Cláusula SQL correspondiente a `WHERE`.
`argsSelecc`	Lista de argumentos utilizados en el parámetro `seleccion`.

Si quisiéramos eliminar un solo elemento, podríamos obtener su URI e indicarlo en el primer parámetro, dejando los otros dos a `null`. Si por el contrario quieres eliminar varios elementos, puedes utilizar el parámetro `seleccion`. Por ejemplo, si quisiéramos eliminar todos los registros de llamada del número 555555555, escribiríamos:

```
getContentResolver().delete(Calls.CONTENT_URI, "number='555555555'", null)
```

También puedes utilizar el método `update()` para modificar elementos de un ContentProvider:

```
int ContentProvider.update(Uri uri, ContentValues valores,
                           String seleccion, String[] argsSelecc)
```

Por ejemplo, si quisiéramos modificar los registros con número 555555555 por el número 444444444, escribiríamos:

```
ContentValues valores2 = new ContentValues();
valores2.put(Calls.NUMBER, "444444444");
getContentResolver().update(Calls.CONTENT_URI, valores2,
                                      "number='555555555'", null);
```

9.8.3. Creación de un ContentProvider

NOTA: Se trata de un aspecto avanzado no necesario en la mayoría de aplicaciones.

En este apartado vamos a describir los pasos necesarios para crear tu propio ContentProvider. En concreto, vamos a seguir tres pasos:

1. Definir una estructura de almacenamiento para los datos. En nuestro caso usaremos una base de datos SQLite.
2. Crear nuestra clase extendiendo `ContentProvider`.
3. Declarar el ContentProvider en el `AndroidManifest.xml` de nuestra aplicación.

Si no deseas compartir tu información con otras aplicaciones, no es necesario crear un ContentProvider. En ese caso resulta mucho más sencillo usar una base de datos directamente, tal y como se ha explicado en el apartado anterior. En este apartado vamos a seguir el mismo ejemplo descrito en los anteriores apartados del capítulo, es decir, vamos a crear un ContentProvider que nos permita compartir la lista de puntuaciones con otras aplicaciones. Posiblemente, no se trate de una situación muy realista. Es de suponer que ninguna aplicación estará interesada en esta información. No obstante, por razones didácticas resulta más sencillo continuar con el mismo ejemplo.

Crea una nueva aplicación que se llame PuntuacionesProvider y cuyo nombre de paquete sea `org.example.puntuacionesprovider`.

Definir la estructura de almacenamiento del ContentProvider

El objetivo último de un ContentProvider es almacenar información de forma permanente. Por lo tanto, resulta imprescindible utilizar alguno de los mecanismos descritos en este tema, o en el siguiente, para almacenar datos. Como se ha estudiado en el apartado anterior, podemos realizar consultas a un ContentProvider de forma similar a una base de datos (podemos hacer consultas SQL y nos devuelve un objeto de tipo `Cursor`). Por lo tanto, la forma más sencilla de almacenar los datos de un ContentProvider es en una base de datos. De esta forma, si nos solicitan una consulta SQL, no tendremos más que trasladarla a nuestra base de datos, y el objeto `Cursor` que nos devuelva será el resultado que nosotros devolveremos.

Para crear la base de datos de nuestro ContentProvider, añade una nueva clase que se llame `PuntuacionesSQLiteHelper` e introduce el siguiente código:

```
public class PuntuacionesSQLiteHelper extends SQLiteOpenHelper {

   public PuntuacionesSQLiteHelper(Context context) {
      super(context, "puntuaciones", null, 1);
   }
```

```
    @Override public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE puntuaciones ("
                + "_id INTEGER PRIMARY KEY AUTOINCREMENT, "
                + "puntos INTEGER, "
                + "nombre TEXT, "
                + "fecha BIGINT)");
    }
    @Override public void onUpgrade(SQLiteDatabase db, int oldVersion,
                                                    int newVersion) {
    // En caso de una nueva versión habría que actualizar las tablas
    }
}
```

La clase que acabamos de añadir al proyecto se encarga de crear la base de datos `puntuaciones` extendiendo la clase `SQLiteOpenHelper`. Este proceso se ha descrito en el apartado dedicado a las bases de datos. Dado que estamos trabajando sobre el mismo ejemplo, la tabla creada es idéntica y lo mismo ocurre con el código. Para una explicación más detallada recomendamos consultar dicho apartado.

Extendiendo la clase ContentProvider

Ahora abordamos la parte más laboriosa: la creación de una clase descendiente de `ContentProvider`.

Los métodos principales que tenemos que implementar son:

`getType()` – devuelve el tipo MIME de los elementos del ContentProvider.

`query()` – permite realizar consultas al ContentProvider.

`insert()` – inserta nuevos datos.

`delete()` – borra elementos del ContentProvider.

`update()` – permite modificar los datos existentes.

La clase `ContentProvider` es *thread-safe*, es decir, toma las precauciones necesarias para evitar problemas con las llamadas simultáneas de varios procesos. Por lo tanto, en la creación de una subclase no nos tenemos que preocupar de este aspecto.

Añade una nueva clase que se llame `PuntuacionesProvider` e introduce el siguiente código:

```
public class PuntuacionesProvider extends ContentProvider {
    public static final String AUTORIDAD = "org.example.puntuacionesprovider";
    public static final Uri CONTENT_URI = Uri.parse("content://"
                              + AUTORIDAD   + "/puntuaciones");
    public static final int TODOS_LOS_ELEMENTOS = 1;
    public static final int UN_ELEMENTO = 2;
    private static UriMatcher URI_MATCHER = null;
    static {
        URI_MATCHER = new UriMatcher(UriMatcher.NO_MATCH);
        URI_MATCHER.addURI(AUTORIDAD, "puntuaciones", TODOS_LOS_ELEMENTOS);
        URI_MATCHER.addURI(AUTORIDAD, "puntuaciones/#", UN_ELEMENTO);
    }
```

```
public static final String TABLA = "puntuaciones";
private SQLiteDatabase baseDeDatos;

@Override   public boolean onCreate() {
   PuntuacionesSQLiteHelper dbHelper = new
                 PuntuacionesSQLiteHelper(getContext());
   baseDeDatos = dbHelper.getWritableDatabase();
   return baseDeDatos != null && baseDeDatos.isOpen();
}
```

Como no podría ser de otra forma, la clase extiende `ContentProvider`. A continuación creamos constantes, `AUTORIDAD` y `CONTENT_URI`, que identificaran nuestro ContentProvider mediante la URI:

```
content://org.example.puntuacionesprovider/puntuaciones
```

Las siguientes líneas permiten crear el objeto estático `URI_MATCHER` de la clase `UriMatcher`. Es habitual en Java utilizar variables estáticas finales para albergar objetos que no van a cambiar en toda la vida de la aplicación, es decir, que son constantes. Conviene recordar que solo se creará un objeto `URI_MATCHER` aunque se instancie varias veces la clase `PuntuacionesProvider`. La clase `UriMatcher` permite diferenciar entre diferentes tipos de URI que vamos a manipular. En nuestro caso, permitimos dos tipos de URI: acabada en `/puntuaciones`, que identifica todas las puntuaciones almacenadas, y acabada en `/puntuaciones/#`, donde hay que reemplazar # por un código numérico que coincida con el campo `_id` de nuestra tabla. Cada tipo de URI ha de tener un código numérico asociado, en nuestro caso `NO_MATCH` (-1), `TODOS_LOS_ELEMENTOS` (1) y `UN_ELEMENTO` (2).

La declaración de variables termina con la constante `TABLA`, que identifica la tabla que gastaremos para almacenar la información y el objeto `baseDeDatos` donde se almacenará la información.

A continuación está el método `onCreate()`, que se llama cuando se crea una instancia de esta clase. Básicamente se crea un `SQLiteHelper` a partir de la clase descrita en el apartado anterior (`PuntuacionesSQLiteHelper`) y se asigna la base de datos resultante a la variable `baseDeDatos`. Devolvemos `true` solo en el caso de que no haya habido problemas en su creación.

Veamos la implementación del método `getType()`, que a partir de una URI nos devuelve el tipo MIME que le corresponde:

```
@Override public String getType(final Uri uri) {
   switch (URI_MATCHER.match(uri)) {
   case TODOS_LOS_ELEMENTOS:
      return "vnd.android.cursor.dir/vnd.org.example.puntuacion";
   case UN_ELEMENTO:
      return "vnd.android.cursor.item/vnd.org.example.puntuacion";
   default:
      throw new IllegalArgumentException("URI incorrecta: " + uri);
   }
}
```

NOTA: Los tipos MIME (Multipurpose Internet Mail Extensions) fueron creados para identificar un tipo de datos concreto que añadíamos como anexo a un correo electrónico, aunque en la actualidad son utilizados por muchos sistemas y protocolos. Un tipo MIME tiene dos partes: `tipo_genérico/tipo_específico`; por ejemplo, `image/gif`, `image/jpeg`, `text/html`, etc.

Existe un convenio para identificar los tipos MIME que proporciona un ContentProvider. Si se trata de un recurso único, utilizamos:

```
vnd.android.cursor.item/vnd.ELEMENTO
```

Y si se trata de una colección de recursos, utilizamos:

```
vnd.android.cursor.dir/vnd.ELEMENTO
```

Donde `ELEMENTO` ha de ser reemplazado por un identificador que describa el tipo de datos. En nuestro caso hemos elegido `org.example.puntuacion`. Utilizar prefijos para definir el elemento minimiza el riesgo de confusión con otro ya existente.

A continuación hemos de sobrescribir los métodos `query`, `insert`, `delete` y `update`, que permitan consultar, insertar, borrar y actualizar elementos de nuestro ContentProvider. Comencemos por el primer método:

```
@Override public Cursor query(Uri uri, String[] proyeccion, String
            seleccion, String[] argSeleccion, String orden) {
   SQLiteQueryBuilder queryBuilder = new SQLiteQueryBuilder();
   queryBuilder.setTables(TABLA);
   switch (URI_MATCHER.match(uri)) {
   case TODOS_LOS_ELEMENTOS:
      break;
   case UN_ELEMENTO:
      String id = uri.getPathSegments().get(1);
      queryBuilder.appendWhere("_id = " + id);
      break;
   default:
      throw new IllegalArgumentException("URI incorrecta "+uri);
   }
   return queryBuilder.query(baseDeDatos, proyeccion, seleccion,
         argSeleccion, null, null, orden);
}
```

El método `query()` que hemos de implementar tiene parámetros similares a `SQLiteQueryBuilder.query()`. Por lo tanto, no tenemos más que trasladar la consulta a nuestra base de datos. No obstante, existe un pequeño inconveniente si nos pasan una URI que identifique un solo elemento, como la mostrada a continuación:

```
content://org.example.puntuacionesprovider/puntuaciones/324
```

Hemos de asegurarnos que solo se devuelve el elemento con `_id` = 324. Para conseguirlo se introduce una cláusula `switch`, que en caso de tratarse de una URI de tipo `UN_ELEMENTO`, añada a la cláusula `WHERE` de la consulta la condición correspondiente mediante el método `appendWhere()`. La cláusula `WHERE` puede tener más condiciones si se ha utilizado el parámetro `seleccion`.

```
@Override public Uri insert(Uri uri, ContentValues valores) {
   long IdFila = baseDeDatos.insert(TABLA, null, valores);
   if (IdFila > 0) {
      return ContentUris.withAppendedId(CONTENT_URI, IdFila);
   } else {
      throw new SQLException("Error al insertar registro en "+uri);
   }
}
@Override
public int delete(Uri uri, String seleccion, String[] argSeleccion) {
   switch (URI_MATCHER.match(uri)) {
   case TODOS_LOS_ELEMENTOS:
      break;
   case UN_ELEMENTO:
      String id = uri.getPathSegments().get(1);
      if (TextUtils.isEmpty(seleccion)) {
         seleccion = "_id = " + id;
      } else {
         seleccion = "_id = " + id + " AND (" + seleccion + ")";
      }
      break;
   default:
      throw new IllegalArgumentException("URI incorrecta: " + uri);
   }
   return baseDeDatos.delete(TABLA, seleccion, argSeleccion);
}
```

El método `insert()` no requiere explicaciones adicionales.

El método `delete()`, igual como ocurrió con el método `query()`, presenta el inconveniente de que pueden habernos indicado una URI que identifique un solo elemento (.../`puntuaciones/324`). En el método `query()` solucionamos este problema llamando a `SQLiteQueryBuilder.appendWhere()`. Sin embargo, ahora no disponemos de un objeto de esta clase, por lo que nos vemos obligados a realizar este trabajo a mano. En caso de no haberse indicado nada en `seleccion`, este parámetro valdrá "`_id = 324`"; y en caso de haberse introducido una condición, por ejemplo "`numero = '555'`", `seleccion`, valdrá "`_id = 324 AND (numero = '555')`".

```
@Override public int update(Uri uri, ContentValues valores, String
                        seleccion, String[] ArgumentosSeleccion) {
   switch (URI_MATCHER.match(uri)) {
   case TODOS_LOS_ELEMENTOS:
      break;
   case UN_ELEMENTO:
      String id = uri.getPathSegments().get(1);
      if (TextUtils.isEmpty(seleccion)) {
         seleccion = "_id = " + id;
      } else {
         seleccion = "_id = " + id + " AND (" + seleccion + ")";
      }
      break;

   default:
      throw new IllegalArgumentException("URI incorrecta: " + uri);
```

```
    }
    return baseDeDatos.update(TABLA, valores, seleccion,
                                         ArgumentosSeleccion);
} } // fin de la clase
```

Finalizamos con el método `update()`, que es muy similar a `delete()`.

Declarar el ContentProvider en *AndroidManifest.xml*

Si queremos que nuestro `ContentProvider` sea visible para otras aplicaciones, resulta imprescindible hacérselo saber al sistema declarándolo en el `AndroidManifest.xml`. Para conseguirlo no tienes más que añadir el siguiente código dentro de la etiqueta `<application>`:

```
<provider
    android:authorities="org.example.puntuacionesprovider"
    android:name="org.example.puntuacionesprovider.PuntuacionesProvider"
    android:exported="true"/>
```

La declaración de un `ContentProvider` requiere que se especifiquen los atributos:

`name`: nombre cualificado de la clase donde hemos implementado nuestro ContentProvider.

`authorities`: parte correspondiente a la autoridad de las URI que vamos a publicar. Puede indicarse más de una autoridad.

También se pueden indicar otros atributos en la etiqueta `<provider>`. Veamos los más importantes:

`label`: etiqueta que describe el ContentProvider que se mostrará al usuario. Es una referencia a un recurso de tipo *string*.

`icon`: una referencia a un recurso de tipo *drawable* con un icono que represente nuestro ContentProvider.

`enabled`: indica si está habilitado. El valor por defecto es `true`.

`exported`: indica si se puede acceder a él desde otras aplicaciones. El valor por defecto es `false`. Si está en `false` solo podrá ser utilizado por aplicaciones con el mismo *id* de usuario que la aplicación donde se crea.

`readPermission`: permiso requerido para consultar el ContentProvider.

`writePermission`: permiso requerido para modificar el ContentProvider.

`permission`: permiso requerido para consultar o modificar el ContentProvider. Sin efecto si se indica `readPermission` o `writePermission`. Para más información, consúltese el capítulo sobre seguridad.

`multiprocess`: indica si cualquier proceso puede crear una instancia del ContentProvider (`true`) o solo el proceso de la aplicación donde se ha creado (`false`, valor por defecto).

`process`: nombre del proceso en el que el ContentProvider ha de ejecutarse. Habitualmente, todos los componentes de una aplicación se ejecutan en el mismo proceso creado para la aplicación. Si no se indica lo contrario, el

nombre del proceso coincide con el nombre del paquete de la aplicación (si lo deseas puedes cambiar este nombre con el atributo `process` de la etiqueta `<aplication>`). Si prefieres que un componente de la aplicación se ejecute en su propio proceso, has de utilizar este atributo.

`initOrder`: orden en que el ContentProvider ha de ser instalado en relación con otros ContentProvider del mismo proceso.

`syncable`: indica si la información del ContentProvider está sincronizada con un servidor.

9.8.4. Acceso a PuntuacionesProvider desde Asteroides

Una vez hemos creado y declarado nuestro ContentProvider, vamos a probarlo desde la aplicación Asteroides. Como hemos hecho en ejemplos anteriores, vamos a crear una nueva clase que implemente la interfaz `AlmacenPuntuaciones`.

Crea una nueva clase en la aplicación `Asteroides` con el nombre `AlmacenPuntuacionesProvider`. Introduce el siguiente código:

```
public class AlmacenPuntuacionesProvider implements AlmacenPuntuaciones {
   private Activity activity;

   public AlmacenPuntuacionesProvider(Activity activity) {
      this.activity = activity;
   }

   public void guardarPuntuacion(int puntos, String nombre, long fecha) {
      Uri uri = Uri.parse(
            "content://org.example.puntuacionesprovider/puntuaciones");
      ContentValues valores = new ContentValues();
      valores.put("nombre", nombre);
      valores.put("puntos", puntos);
      valores.put("fecha", fecha);
      try {
         activity.getContentResolver().insert(uri, valores);
      } catch (Exception e) {
         Toast.makeText(activity, "Verificar que está instalado ”+
           “org.example.puntuacionesprovider",Toast.LENGTH_LONG).show();
         Log.e("Asteroides", "Error: " + e.toString(), e);
      }
   }

   public List<String> listaPuntuaciones(int cantidad) {
      List<String> result = new ArrayList<String>();
      Uri uri = Uri.parse(
            "content://org.example.puntuacionesprovider/puntuaciones");
      Cursor cursor = activity. getContentResolver().query (uri,
                              null, null, null, "fecha DESC");
      if (cursor != null) {
         while (cursor.moveToNext()) {
            String nombre = cursor.getString(
                              cursor.getColumnIndex("nombre"));
            int puntos = cursor.getInt(
                              cursor.getColumnIndex("puntos"));
```

```
            result.add(puntos + " " + nombre);
        }
      }
      return result;
   }
}
```

Modifica el código correspondiente para que la nueva clase pueda ser seleccionada como almacén de las puntuaciones. Recuerda que has de instalar primero la aplicación `PuntuacionesProvider` para que funcione este ejemplo.



Preguntas de repaso: *ContentProvider*

CAPÍTULO 10.
Internet: *sockets*, HTTP y servicios web

Los teléfonos Android suelen disponer de conexión a Internet. Esto nos permite no solo almacenar los datos en nuestro dispositivo, si no también compartirlos con otros usuarios. En el primer punto del capítulo trataremos de resolver el problema de comunicar dos aplicaciones en Internet mediante la herramienta básica: los *sockets.* Existen otras alternativas de más alto nivel, como el uso del protocolo HTTP, que se estudiará en el segundo punto del capítulo. En el tercer punto se tratará una tercera alternativa, todavía de más alto nivel: los servicios web.

En este capítulo implementaremos el mismo ejemplo que en el capítulo anterior, es decir, almacenaremos las puntuaciones obtenidas en Asteroides, pero ahora en un servidor de Internet. Utilizaremos las tres alternativas descritas en el párrafo anterior. No obstante, has de tener claro que estos mecanismos están relacionados entre sí. Por ejemplo, si utilizas servicios web, internamente se utilizará el protocolo HTTP, y además este protocolo utiliza *sockets* para establecer la comunicación.

Realizar peticiones HTTP conlleva un uso adecuado de hilos y posiblemente el almacenamiento de los datos en caché. Como se trata de una tarea frecuente, puede ser interesante apoyarse en una librería que automatice este trabajo. Terminaremos el capítulo describiendo el uso de la librería Volley.



Objetivos:

- Repasar las alternativas para intercambiar datos a través de Internet.
- Describir el uso de *sockets*, como herramienta básica para comunicar aplicaciones por Internet.
- Mostrar cómo programar un protocolo basado en *sockets* sobre TCP.
- Describir el funcionamiento del protocolo HTTP
- Mostrar cómo programar peticiones HTTP desde Android.
- Definir el concepto de servicio web, en las alternativas SOAP y REST.

- Mostrar el acceso a servicios web de terceros desde Android.
- Aprender a crear nuestro propio servidor de servicios web con PHP, Apache y MySQL.
- Realizar peticiones HTTP de forma sencilla usando la librería Volley.

10.1. Comunicaciones en Internet mediante *sockets*

Antes de definir qué es un *socket* conviene aclarar los roles o configuraciones que pueden tomar las aplicaciones en un proceso de comunicación. Las dos configuraciones más importantes que pueden tomar son: las llamadas "arquitectura igual a igual" y la "arquitectura cliente/servidor". Esta segunda es la que utilizaremos en los ejemplos de este capítulo; por tanto, conviene aclarar este aspecto.

10.1.1. La arquitectura cliente/servidor

Las aplicaciones en Internet suelen seguir la arquitectura cliente/servidor. Esta arquitectura se caracteriza por descomponer el trabajo en dos partes (es decir, dos programas): el servidor, que centraliza el servicio, y el cliente, que controla la interacción con el usuario. El servidor ha de ofrecer sus servicios a través de una dirección conocida. Algunos ejemplos de aplicaciones basadas en la arquitectura cliente/servidor son WWW o el correo electrónico. En esta arquitectura se suelen seguir las siguientes pautas de comportamiento:

Cliente	Servidor
1. Se conecta al servidor.	1. A la espera de que algún cliente se conecte.
2. Solicita alguna información al servidor.	2. Recibe solicitud.
3. Recibe la respuesta.	3. Envía respuesta.
4. Ir al punto 2, si hay más solicitudes.	4. Ir al punto 2, si hay más solicitudes.
5. Cierra la conexión.	5. Cierra la conexión.
	6- Ir al punto 1

10.1.2. ¿Qué es un *socket*?

Cada una de las diferentes aplicaciones en Internet (web, correo electrónico, etc.) ha de poder intercambiar información entre programas situados en diferentes ordenadores o dispositivos. Con este propósito, se va a hacer uso del nivel de transporte de la pila de protocolos TCP/IP, cuyo objetivo final es permitir el intercambio de información a través de la red de forma fiable y transparente.



Vídeo[tutorial]: *La interfaz* socket

La interfaz *socket* define las reglas que un programa ha de seguir para utilizar los servicios del nivel de transporte en una red TCP/IP. Esta interfaz se basa en el concepto de *socket*. Un *socket* es el punto final de una comunicación bidireccional entre dos programas que intercambian información a través de Internet (*socket* se traduce literalmente como "enchufe").

Dado que en un mismo dispositivo/ordenador podemos estar ejecutando de forma simultánea diferentes aplicaciones que utilizan Internet para comunicarse, resulta imprescindible identificar cada *socket* con una dirección diferente. Un *socket* se va a identificar por la dirección IP del dispositivo, más un número de puerto (de 16 bits). En Internet se suele asociar a cada aplicación un número de puerto concreto (por ejemplo: 80 para la web, 25 para el correo electrónico, 7 para ECHO o 4661 para eDonkey).

Una conexión está determinada por un par de *sockets*, uno en cada extremo de la conexión. Existen dos tipos de *socket*: *socket stream* y *socket datagram*. Veamos en qué se diferencian:

Sockets stream (TCP)

Los *sockets stream* ofrecen un servicio orientado a la conexión, donde los datos se transfieren como un flujo continuo, sin encuadrarlos en registros o bloques. Este tipo de *socket* se basa en el protocolo TCP, que es un protocolo orientado a la conexión. Esto implica que antes de transmitir información hay que establecer una conexión entre los dos *sockets*. Mientras uno de los *sockets* atiende peticiones de conexión (servidor), el otro solicita la conexión (cliente). Una vez que los dos *sockets* están conectados, ya se puede transmitir datos en ambas direcciones. El protocolo incorpora de forma transparente al programador la corrección de errores. Es decir, si detecta que parte de la información no ha llegado a su destino correctamente, esta volverá a trasmitirse. Además, no limita el tamaño máximo de información a transmitir.

Sockets datagram (UDP)

Los *sockets datagram* se basan en el protocolo UDP y ofrecen un servicio de transporte sin conexión. Es decir, podemos mandar información a un destino sin necesidad de realizar una conexión previa. El protocolo UDP es más eficiente que el TCP, pero tiene el inconveniente de que no se garantiza la fiabilidad. Además, los datos se envían y reciben en datagramas (paquetes de información) de tamaño limitado. La entrega de un datagrama no está garantizada: estos pueden duplicarse, perderse o llegar en un orden diferente del que se envió.

La gran ventaja de este tipo de *sockets* es que apenas introducen sobrecarga sobre la información transmitida. Además, los retrasos introducidos son mínimos, lo cual los hace especialmente interesantes para aplicaciones en tiempo real, como la transmisión de audio y vídeo sobre Internet. Sin embargo, presentan muchos inconvenientes para el programador: cuando transmitimos un datagrama no tenemos la certeza de que este llegue a su destino, por lo que, si fuera necesario, tendríamos que implementar nuestro propio mecanismo de control de errores. Otro inconveniente es el hecho de que existe un tamaño máximo de datagrama: unos 1500 bytes dependiendo de la implementación. Si la información a enviar es mayor,

tendremos que fraccionarla y enviar varios datagramas independientes. En el destino tendremos que concatenarlos en el orden correcto.

En conclusión, si deseas una comunicación libre de errores y sin preocupaciones para el programador, es más conveniente que utilices *sockets stream*. Es el tipo de *sockets* que utilizaremos en los siguientes ejemplos.

10.1.3. Un ejemplo de un cliente/servidor de ECHO

El servicio ECHO suele estar instalado en el puerto 7 de máquinas Unix y permite comprobar que la máquina está operativa y que se puede establecer una conexión con dicha máquina.

El funcionamiento de un servidor ECHO es muy sencillo: cuando alguien se conecta espera que el servidor le envíe algo y le responde exactamente con la misma información recibida. El cliente actúa de forma contraria: envía datos al servidor y luego comprueba que los datos recibidos son idénticos a los transmitidos.



Ejercicio: *Estudio del protocolo ECHO con el comando Telnet*

El siguiente ejercicio nos muestra un truco para testear si un servidor que utiliza TCP funciona correctamente. Te recomendamos que lo utilices antes de realizar la programación del cliente. Por una parte, te permitirá asegurarte de que tanto la conexión como el servidor funcionan correctamente. Por otra parte, te asegurarás de que has entendido correctamente el protocolo a implementar.

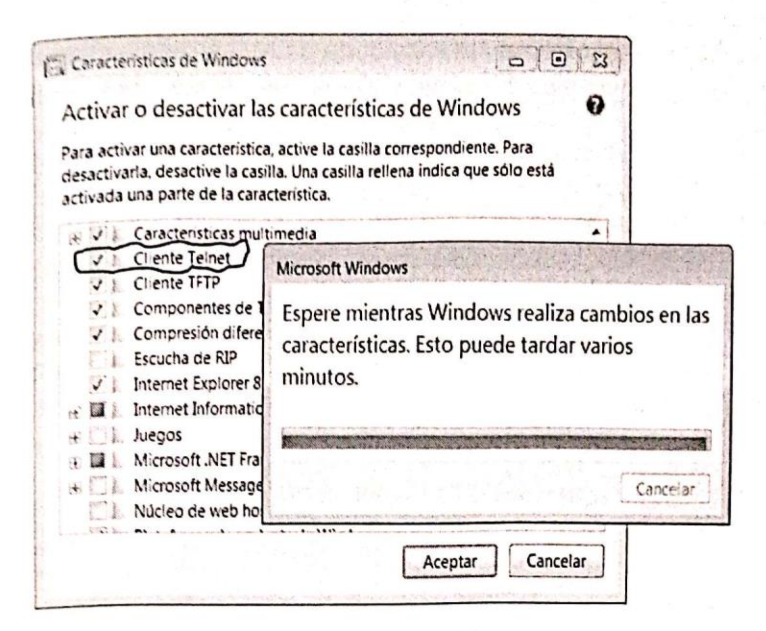
> ***NOTA:*** *Para que el ejemplo funcione en la dirección IP* `158.42.146.127` *ha de haber un servidor de ECHO en funcionamiento. En caso de no ser así, puedes reemplazar la IP por la de un servidor de ECHO en funcionamiento. También puedes implementar tu propio servidor de ECHO, tal y como se muestra en uno de los siguientes ejercicios.*

1. Para verificar si el servidor de ECHO está en marcha, desde un intérprete de comandos (símbolo del sistema/*shell*) escribe:

   ```
   telnet 158.42.146.127 7
   ```

   Este comando permite establecer una conexión TCP con el puerto 7 del servidor. A partir de ahora todo lo que escribas se enviará al servidor (aunque en muchos clientes Telnet no se muestra en pantalla lo que escribes) y todo lo que el servidor envíe se imprimirá en pantalla.

   > ***NOTA:*** *Windows 7 tiene desactivado por defecto el comando Telnet. Para habilitarlo, haz clic en el menú Inicio > Panel de control > Programas > Activar o desactivar las características de Windows y marca Cliente Telnet.*



2. Espera a que se establezca la conexión. De no ser posible, vuelve a intentarlo o lee la nota del principio del ejercicio.

3. Escribe una frase cualquiera y pulsa <Intro>. Por ejemplo:

```
Hola hola↵
```

   Si te equivocas, no uses la tecla de borrar. Equivocarse en este protocolo no tiene ninguna repercusión, pero en el resto de lo que vamos a estudiar hará que el servidor no nos entienda.

4. Observa que la respuesta obtenida coincide con la frase que has introducido. Además, el servidor cerrará inmediatamente la conexión y no te permitirá mandar más frases.



Ejercicio: *Un cliente de ECHO*

El siguiente ejemplo muestra cómo podrías desarrollar un cliente de ECHO que utiliza un *socket stream* desde Android.

> ***NOTA:*** *Para que el ejemplo funcione en la dirección IP* `158.42.146.127` *ha de haber un servidor de ECHO en funcionamiento. En caso de no ser así, puedes reemplazar la IP por la de un servidor de ECHO en funcionamiento o realizar el siguiente ejercicio.*

1. Crea una nueva aplicación con los siguientes datos:

```
Template / Phone and Tablet / Empty Activity
Application Name: Cliente ECHO
Minimum SDK: API 15 Android 4.0.3 (IceCreamSandwich)
```

2. Añade la etiqueta <android:id="@+id/TextView01"> en el TextView del *layout* de la actividad.

3. Reemplaza el código de la actividad por:

```
public class MainActivity extends Activity {
   private TextView output;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      output = findViewById(R.id.TextView01);
      StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()
            .permitNetwork().build());
      ejecutaCliente();
   }

   private void ejecutaCliente() {
      String ip = "158.42.146.127";
      int puerto = 7;
      log(" socket " + ip + " " + puerto);
      try {
         Socket sk = new Socket(ip, puerto);
         BufferedReader entrada = new BufferedReader(
                  new InputStreamReader(sk.getInputStream()));
         PrintWriter salida = new PrintWriter(
                  new OutputStreamWriter(sk.getOutputStream()), true);
         log("enviando... Hola Mundo ");
         salida.println("Hola Mundo");
         log("recibiendo ... " + entrada.readLine());
         sk.close();
      } catch (Exception e) {
         log("error: " + e.toString());
      }
   }

   private void log(String string) {
      output.append(string + "\n");
   }
}
```

Las tres primeras líneas del método `onCreate()` han de resultarte familiares. En la cuarta se configura StrictMode[43]. Consiste en una herramienta de desarrollo que detecta cosas que podrías estar haciendo mal y te llama la atención para que las corrijas. Esta herramienta aparece en el nivel de API 9. Una de las verificaciones que realiza es que no se acceda a la red desde el hilo principal. Este problema se podría resolver lanzando un nuevo hilo para realizar el acceso al servidor[44]. Más adelante se muestra cómo resolverlo usando `AsyncTask`. En este apartado queremos centrarnos en el uso de *sockets* y no en el manejo de hilos, por lo que vamos a desactivar esta comprobación. En la línea en cuestión se indica a StrictMode que en su política de *threads* no tenga en cuenta los accesos a la red.

---

[43] http://developer.android.com/reference/android/os/StrictMode.html

[44] En el capítulo 5 se describe este problema con más detalle.

La parte interesante se encuentra en el método `ejecutarCliente()`. En primer lugar, todo cliente ha de conocer la dirección del *socket* del servidor; en este caso los valores se indican en el par de variables `ip` y `puerto`. Nunca tenemos la certeza de que el servidor admita la conexión, por lo que es obligatorio utilizar una sección `try`/`catch`. La conexión propiamente dicha se realiza con el constructor de la clase `Socket`. Siempre hay que tener previsto que ocurra algún problema. En tal caso, se creará la excepción y se pasará a la sección `catch`. En caso contrario, continuaremos obteniendo el `InputStream` y el `OutputStream` asociado al *socket*, lo cual nos permitirá obtener las variables `entrada` y `salida`, mediante las que podremos recibir y transmitir información. El programa transmite la cadena `"Hola Mundo"`, tras lo que visualiza la información recibida. Si todo es correcto, ha de coincidir con lo transmitido.

4. Solicita en la aplicación el permiso `INTERNET` en *AndroidManifest*.

```
<uses-permission android:name="android.permission.INTERNET"/>
```

5. Ejecuta la aplicación y verifica si el servidor responde. Recuerda que es posible que no se haya arrancado este servicio en el servidor.

6. Comenta la línea de `onCreate()` donde se configura `StrictMode` y ejecuta de nuevo la aplicación. El resultado no será satisfactorio. En este caso ocurrirá una excepción `NetworkOnMainThreadException`.



Ejercicio: *Un servidor de ECHO*

Vamos a implementar un servidor de ECHO en tu ordenador personal. Has de tener claro que no va a ser una aplicación Android, si no un programa 100 % Java.

1. Crea un nuevo proyecto Java y llámalo *ServidorECHO*.

   Crea un nuevo proyecto (*File > New Project...*). Utiliza la opción *Add No Activity*, así no creamos una actividad que nunca será usada. Pulsa en *File > New Module*. Selecciona *Java Library* y pulsa *Next*. Introduce en *Library name:* `servidor`, como *Java pakage name:* `com.example.servidorecho` y en *Java class name:* `ServidorECHO`. Pulsa el botón *Finish*. Se creará un nuevo módulo Java dentro de tu proyecto Android. Pulsa en el botón desplegable a la derecha del botón *Run* app ▸. Selecciona *Edit Configurations...* En la nueva ventana, haz clic en el signo + de la esquina superior izquierda y selecciona *Application*. Aparecerá una nueva configuración de aplicación. Selecciona en *Name:* `servidor`, en *Main class:* `com.example.servidorecho.ServidorECHO` y en *Use classpath of module:* `servidor`. Pulsa en *OK*.

2. Reemplaza el código de `ServidorECHO` por el siguiente código:

```
public class ServidorECHO {
   public static void main(String args[]) {
      try {
         System.out.println("Servidor en marcha...");
```

```
         ServerSocket sk = new ServerSocket(7);
         while (true) {
            Socket cliente = sk.accept();
            BufferedReader entrada = new BufferedReader(
                  new InputStreamReader(cliente.getInputStream()));
            PrintWriter salida = new PrintWriter(new OutputStreamWriter(
                  cliente.getOutputStream()), true);
            String datos = entrada.readLine();
            salida.println(datos);
            cliente.close();
            System.out.println(datos);
         }
      } catch (IOException e) {
         System.out.println(e);
      }
   }
}
```

En este caso utilizaremos la clase `ServerSocket` asociada al puerto 7 para crear un *socket* que acepta conexiones. Luego se introduce un bucle infinito para que el servidor esté perpetuamente en servicio. El método `accept()` bloquea al servidor hasta que un cliente se conecte. Cuando ocurra esto, todo el intercambio de información se realizará a través de un nuevo *socket* creado con este propósito, el *socket* `cliente`. El resto del código es similar al cliente, aunque en este caso primero se recibe y luego se transmite lo mismo que se ha recibido.

3. Sustituye la dirección IP en ClienteECHO por la IP de tu ordenador. El comando `ipconfig` (Windows) o `ifconfig` (Linux/Mac) te permite averiguar la dirección IP de tu ordenador. No utilices como IP `127.0.0.1` (*localhost*), dado que, aunque se ejecuten en la misma máquina, la IP del emulador es diferente de la del PC.
4. Ejecuta primero el servidor y luego el cliente para verificar que funciona.

NOTA: Si la IP de tu ordenador es privada, no podrás crear un servidor accesible desde cualquier parte de Internet. En este caso utiliza para el cliente un emulador o un dispositivo Android real que se conecte por Wi-Fi a la misma red de tu ordenador. De lo contrario, el cliente no encontrará el servidor.

10.1.4. Un servidor por *sockets* para las puntuaciones

Siguiendo la estructura básica de un cliente y un servidor TCP que acabamos de ver, resultará muy sencillo implementar un protocolo que permita a varios clientes conectarse a un servidor para consultar la lista de puntuaciones o mandar nuevas puntuaciones. El primer lugar, tenemos que diseñar un protocolo que permita realizar estas dos operaciones.

Para consultar puntuaciones, el cliente se conectará al servidor y le mandará los caracteres `PUNTUACIONES` (solo se permite en mayúsculas) seguidos de un salto de línea. El servidor mandará toda la lista de puntuaciones, separadas por caracteres de salto de línea. A continuación se cerrará la conexión.

Cliente: `PUNTUACIONES↵`

Servidor: `19000 Pedro Perez↵`

```
17500 María Suarez↵
13000 Juan García↵
```

Para almacenar una nueva puntuación, el cliente se conectará al servidor y mandará un texto con la puntuación obtenida, seguido de un salto de línea. El servidor lo reconocerá como una nueva puntuación siempre que este texto no sea `PUNTUACIONES`. En tal caso almacenará la puntuación y mandará los caracteres `OK` seguidos de un salto de línea. A continuación se cerrará la conexión.

Cliente: `32000 Eva Gutierrez↵`

Servidor: `OK↵`

En segundo lugar, hay que elegir un número de puerto para realizar la comunicación; por ejemplo, el 1234.



Ejercicio: *Estudio del protocolo PUNTUACIONES con el comando Telnet*

El siguiente ejercicio nos muestra un truco para testear si un servidor que utiliza TCP funciona correctamente. Te recomendamos que lo utilices antes de realizar la programación del cliente. Por una parte, te permitirá asegurarte de que tanto la conexión como el servidor funcionan correctamente. Por otra parte, te asegurarás de que has entendido correctamente el protocolo a implementar.

> ***NOTA:*** *Para que el ejemplo funcione en la dirección IP* `158.42.146.127` *ha de haber un servidor de PUNTUACIONES en funcionamiento. En caso de no ser así, implementa tu propio servidor de PUNTUACIONES, tal y como se muestra en uno de los siguientes ejercicios.*

1. Para conectarte al servidor, desde un intérprete de comandos (símbolo del sistema/*shell*) escribe:

   ```
   telnet 158.42.146.127 1234
   ```
2. Si se establece la conexión, escribe un número seguido de tu nombre y pulsa <Intro>. Por ejemplo:

   ```
   14000 Juan García↵
   ```
3. La respuesta obtenida ha de ser `OK`, y luego se cerrará la conexión.
4. Repite el punto 2 y, tras la conexión, escribe:

   ```
   PUNTUACIONES↵
   ```
5. La respuesta obtenida ha de ser la lista de puntuaciones, donde ha de estar la que acabas de introducir.



Ejercicio: *Almacenando las puntuaciones mediante un protocolo basado en* sockets

1. Crea un nuevo proyecto Java (Java SE, no para Android) y llámalo *ServidorPuntuacionesSocket*. Este proyecto ha de contener la clase

`ServidorPuntuaciones`. Instrucciones detalladas se muestran en el ejercicio *Un servidor de ECHO*.

2. Reemplaza el código de la clase por el que se muestra a continuación:

```
public class ServidorPuntuaciones {

      public static void main(String args[]) {
      List<String> puntuaciones = new ArrayList<String>();

      try {
         ServerSocket s = new ServerSocket(1234);
         System.out.println("Esperando conexiones...");

         while (true) {
            Socket cliente = s.accept();
            BufferedReader entrada = new BufferedReader(
                  new InputStreamReader(cliente.getInputStream()));
            PrintWriter salida = new PrintWriter(new OutputStreamWriter(
                  cliente.getOutputStream()), true);
            String datos = entrada.readLine();
            if (datos.equals("PUNTUACIONES")) {
               for (int n = 0; n < puntuaciones.size(); n++) {
                  salida.println(puntuaciones.get(n));
               }
            } else {
               puntuaciones.add(0, datos);
               salida.println("OK");
            }
            cliente.close();
         }
      } catch (IOException e) {
         System.out.println(e);
      }
   }
}
```

3. Ejecuta el proyecto.

4. Verifica que en la vista *Run* aparece: "Esperando conexiones...".

5. Desde la vista *Run* podrás detener la aplicación pulsando el cuadro rojo.

NOTA: Si ejecutas de nuevo la aplicación sin pararla primero, dará un error. Esto es debido a que la aplicación ya lanzada no es detenida y esta aplicación tiene asociado el puerto 1234. El sistema no permitirá que una nueva aplicación escuche este puerto.

6. Abre el proyecto Asteroides y crea la siguiente clase:

```
public class AlmacenPuntuacionesSocket implements AlmacenPuntuaciones{

  public AlmacenPuntuacionesSocket() {
    StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.
            Builder().permitNetwork().build());
  }

  public void guardarPuntuacion(int puntos, String nombre, long fecha){
   try {
```

```
      Socket sk = new Socket("X.X.X.X", 1234);
      BufferedReader entrada = new BufferedReader(
                new InputStreamReader(sk.getInputStream()));
      PrintWriter salida = new PrintWriter(
                new OutputStreamWriter(sk.getOutputStream()),true);
      salida.println(puntos + " " + nombre);
      String respuesta = entrada.readLine();
      if (!respuesta.equals("OK")) {
         Log.e("Asteroides", "Error: respuesta de servidor incorrecta");
      }
      sk.close();
   } catch (Exception e) {
      Log.e("Asteroides", e.toString(), e);
   }
  }

  public List<String> listaPuntuaciones(int cantidad) {
   List<String> result = new ArrayList<String>();
   try {
      Socket sk = new Socket("X.X.X.X", 1234);
      BufferedReader entrada =    new BufferedReader(
             new InputStreamReader(sk.getInputStream()));
      PrintWriter salida = new PrintWriter(
                new OutputStreamWriter(sk.getOutputStream()),true);
      salida.println("PUNTUACIONES");
      int n = 0;
      String respuesta;
      do {
         respuesta = entrada.readLine();
         if (respuesta != null) {
            result.add(respuesta);
            n++;
         }
      } while (n < cantidad && respuesta != null);
      sk.close();
   } catch (Exception e) {
      Log.e("Asteroides", e.toString(), e);
   }
   return result;
  }
}
```

7. Sustituye las dos apariciones de "X.X.X.X" por la dirección IP donde esté ejecutándose el servidor.

   NOTA: *El comando* `ipconfig` *(Windows)* o `ifconfig` *(Linux/Mac) te permite averiguar la dirección IP de tu ordenador. No utilices como IP 127.0.0.1 (localhost), dado que, aunque se ejecuten en la misma máquina, la IP del emulador es diferente de la del PC.*

   NOTA: *Si ejecutas el cliente en un emulador, la IP del PC (servidor) es 10.0.2.2.*

8. Recuerda que ahora la aplicación Asteroides necesita el permiso `INTERNET`. Y tiene que ser compilada con una versión mínima 9 (para `StrictMode`).

9. Ejecuta la aplicación y accede a visualizar la lista de puntuaciones. Luego inicia una partida nueva.
10. Verifica que en la vista *Consola* aparecen las consultas al servidor.
11. Para terminar, reemplaza la IP en el cliente por la siguiente "158.42.146.127" para conectarte a un servidor compartido.

    NOTA: Es posible que este servicio no haya sido iniciado.

12. Modifica el código correspondiente para que la nueva clase pueda ser seleccionada como almacén de las puntuaciones.
13. Comprueba si otros usuarios han accedido a este servidor y aparecen sus puntuaciones.



Preguntas de repaso: *La interfaz* socket

10.2. La web y el protocolo HTTP

Dentro del mundo de Internet destaca una aplicación que es, con mucho, la más utilizada: la World Wide Web (WWW), a la que nos referiremos coloquialmente como la web. Su gran éxito se debe a la facilidad de uso, dado que simplifica el acceso a todo tipo de información, y a que esta información es presentada de forma atractiva. Básicamente, la web nos ofrece un servicio de acceso a información distribuida en miles de servidores en todo Internet, que nos permite ir navegando por todo tipo de documentos multimedia gracias a un sencillo sistema de hipervínculos.

Para la comunicación entre los clientes y los servidores de esta aplicación, se emplea el protocolo HTTP (Hypertext Transfer Protocol), que será el objeto de estudio de este apartado.

10.2.1. El protocolo HTTP

HTTP es un sencillo protocolo cliente-servidor que articula los intercambios de información entre los navegadores web y los servidores web. Fue propuesto por Tim Berners-Lee, atendiendo a las necesidades de un sistema global de distribución de información como la World Wide Web. En la web los servidores han de escuchar en el puerto 80, esperando la conexión de algún cliente web.



Vídeo[tutorial]: *El protocolo HTTP*

A continuación, describimos los pasos habituales que se siguen en una interacción del protocolo HTTP/0.9:

1. El usuario quiere acceder a la página http://www.upv.es/dir/pag.html, para lo cual pincha en un enlace de un documento HTML o introduce la página directamente en el campo *Dirección* del navegador web.
2. El navegador averigua la dirección IP de www.upv.es.
3. El navegador establece una conexión con el puerto 80 de esta IP.
4. El navegador envía por esta conexión (↵ carácter de salto de línea):

```
GET /dir/pag.html ↵
```

5. El servidor envía la página a través de la conexión:

```
<HTML> ↵
<HEAD> ↵
<TITLE>Página de ... </TITLE> ↵
...
</HTML>
```

6. El servidor cierra la conexión.

El protocolo HTTP/0.9 repite este proceso cada vez que el navegador necesita un fichero del servidor. Por ejemplo, si se ha bajado un documento HTML en cuyo interior están insertadas cuatro imágenes, el proceso anterior se repite un total de cinco veces, una para el documento HTML y cuatro para las imágenes.

Como ves, se trata de un protocolo sin estado. Cada petición contiene la información necesaria para ser atendida. Si deseamos mantener un estado, tendrá que ser implementado usando algún mecanismo adicional (por ejemplo, las *cookies*).



Ejercicio: *Estudio del protocolo HTTP/0.9 utilizando el comando Telnet*

1. Abre un navegador web y accede a la página:

   http://www.androidcurso.com/images/dcomg/corta.html

   En caso de que el servidor no responda, puedes realizar el ejercicio con cualquier página de otro servidor. El carácter ~ se obtiene pulsando simultáneamente <Alt Gr> y <4>.

2. Visualiza el contenido HTML de la página (menú "Ver/Código fuente", "Herramientas / Ver código fuente", o similar).
3. Desde un intérprete de comandos (símbolo del sistema/*shell*) escribe:

```
telnet www.dcomg.upv.es 80
```

   Este comando permite establecer una conexión TCP con el puerto 80 del servidor. A partir de ahora todo lo que escribas se enviará al servidor (aunque en muchos casos no lo veas en pantalla) y todo lo que el servidor envíe se imprimirá en pantalla.

***NOTA:** Si utilizas un servidor diferente, asegúrate de que soporta la versión HTTP/0.9.*

4. Cuando se establezca la conexión, teclea exactamente:

```
GET /~jtomas/corta.html↵
```

Si te equivocas, no uses la tecla de borrar. En tal caso, repite el ejercicio desde el punto 3.

5. Observa que la respuesta obtenida coincide con el contenido HTML del paso 2.

10.2.2. Versión 1.0 del protocolo HTTP

Con la popularización de la aplicación WWW, pronto se vio la necesidad de ampliar este sencillo protocolo para permitir nuevas funcionalidades. Se definió la versión 1.0 del protocolo, que añadía nuevos métodos (PUT, POST) además de permitir el intercambio de cabeceras entre cliente y servidor.



Vídeo[tutorial]: *El protocolo HTTP v1.0*

A continuación, se muestra un ejemplo de interacción para la versión 1.0:

Cliente:
```
GET /dir/pag.html HTTP/1.0 ↵
User-Agent: Internet Explorer v3.2 ↵
Host: www.upv.es ↵
Accept: text/html, image/gif, image/jpeg ↵
↵                                          <línea en blanco>
```

Servidor:
```
HTTP/1.1 200 OK ↵
Server: Microsoft-IIS/5.0 ↵
Last-Modified: Mon, 25 Feb 2002 15:49:22 GMT ↵
Content-Type: text/html↵
↵                                          <línea en blanco>
<HTML> ↵
<HEAD> ↵
<TITLE>Página de ... </TITLE> ↵
...
</HTML>
```

En esta nueva versión, el navegador se conecta al puerto 80 del servidor y normalmente le envía el comando "GET" seguido de la página que desea obtener. Ahora el navegador añadirá la palabra "HTTP/1.0" para indicar al servidor que quiere utilizar esta nueva versión del protocolo. A continuación, el navegador introducirá un salto de línea seguido, opcionalmente, de alguna cabecera (véase RFC 2616). Una cabecera consta de un identificador de cabecera, seguido de dos puntos y el valor de la cabecera, más un salto de línea (↵). Estas cabeceras permitirán que el navegador indique al servidor ciertos datos que pueden serle de utilidad. Por ejemplo, con el identificador de cabecera "User-Agent:" se puede indicar el tipo y la versión de navegador con el que se realiza la solicitud. "Host:" permite indicar el ordenador donde se ejecuta el servidor. O la cabecera "Accept:", que permite indicar al servidor qué tipo de documentos es capaz de visualizar el navegador en formato MIME. Cuando el navegador ya no quiera insertar más cabeceras, introducirá una

línea en blanco. Es decir, tras el salto de línea de la última cabecera, manda un nuevo salto de línea.

Cuando el servidor lea una línea en blanco, es decir, dos saltos de línea seguidos, sabrá que le ha llegado su turno y tiene que contestar. En esta versión del protocolo el servidor no transmitirá la página solicitada directamente. Antes contesta con una línea indicando: primero la versión más alta que soporta (normalmente "HTTP/1.1"), a continuación un espacio y un código de tres dígitos que informa de si se puede realizar la operación solicitada, finalizando con una frase explicativa sobre este código. Algunos códigos de respuesta posibles son: `200 OK`, `401 no autorizado`, `404 fichero no encontrado`, etc. Tras esta primera línea el servidor podrá enviar alguna cabecera con información sobre el servidor o el documento que va a transmitir. Cuando ya no quiera insertar más cabeceras introducirá una línea en blanco seguida del documento.

Aunque el método GET es el más utilizado, en la versión 1.0 se añaden nuevos métodos. A continuación se incluye una descripción:

GET:	Petición de lectura de un recurso.
POST:	Envío de información asociada a un recurso del servidor.
PUT:	Creación de un nuevo recurso en el servidor.
DELETE:	Eliminación de un recurso.
HEAD:	El servidor solo transmitirá las cabeceras, no la página.



Ejercicio: *Estudio del protocolo HTTP v1.0 utilizando el comando Telnet*

1. Desde un intérprete de comandos (símbolo del sistema/*shell*) escribe:

```
telnet www.dcomg.upv.es 80
```

2. Cuando se establezca la conexión teclea:

```
GET /~jtomas/corta.html HTTP/1.0↵
↵
```

3. Observa que la respuesta obtenida es similar al ejercicio anterior, pero ahora el servidor ha incluido cabeceras. ¿Qué información puedes sacar de estas cabeceras? ¿Por qué has tenido que introducir dos saltos de línea?

4. Repite el ejercicio utilizando el comando HEAD.

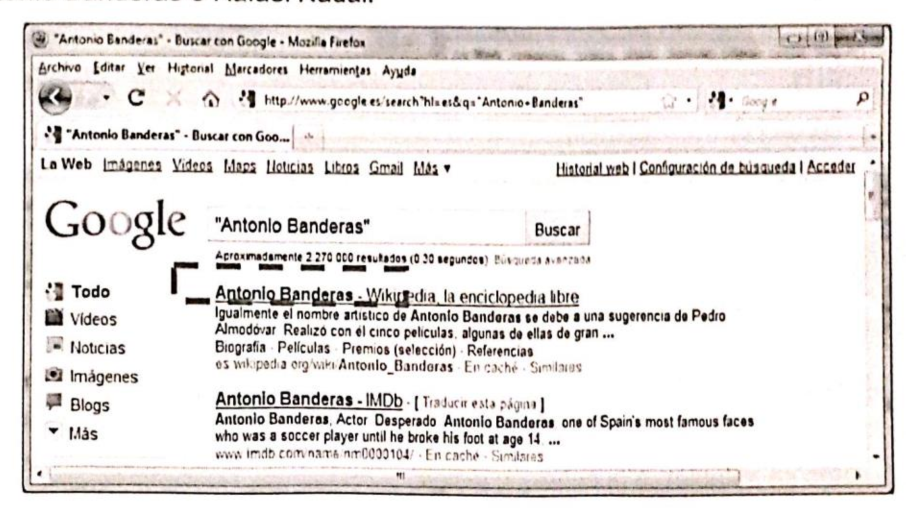
Aunque la versión más reciente es HTTP/1.2 (y existe un borrador de HTTP/2.0), la versión más utilizada en la actualidad es HTTP/1.1. Incorpora algunas mejoras, como las conexiones persistentes, activadas por defecto o la gestión de la caché del cliente. También permite al cliente enviar múltiples peticiones a la vez.

10.2.3. Utilizando HTTP desde Android

Tras el gran éxito de la web, el protocolo HTTP está siendo utilizado con finalidades diferentes de las que tenía en un principio: la descarga de páginas web. Por ejemplo,

hoy en día es frecuente su uso para el intercambio de ficheros, la emisión de vídeo o la comunicación entre aplicaciones. A continuación describiremos las herramientas disponibles en Android para utilizar el protocolo HTTP. Para este propósito tenemos dos alternativas principales desde Android: el uso de las librerías `java.net.*` o `org.apache.commons.httpclient.*`. En el siguiente ejemplo utilizaremos las primeras.

En el ejemplo de este apartado vamos a extraer información de una de las páginas web más utilizadas en la actualidad: el servicio de búsqueda de Google. En concreto, nos interesa conocer el número de apariciones de una determinada secuencia de palabras en la web. En ciertas ocasiones esta información puede resultar muy interesante. Por ejemplo, tenemos dudas sobre el uso de una preposición en inglés: ¿se escribe *travel in bus* o *travel by bus*? Si buscamos ambas secuencias de palabras en Google, hay que ponerlas entre comillas para que busque la secuencia de forma literal. Obtenemos 240.000 apariciones para la primera y 1,1 millones para la segunda. Nuestra duda ha sido resuelta. Esta aplicación también puede utilizarse para averiguar quién es más popular en Internet, Antonio Banderas o Rafael Nadal.



Básicamente, la aplicación que mostramos a continuación funciona de la siguiente forma:

1. El usuario introduce una secuencia de palabras; por ejemplo, "Antonio Banderas".

2. Accedemos al servidor mediante la siguiente URL:

   ```
   http://www.google.es/search?hl=es&q="Antonio+Banderas"
   ```

   En este caso las peticiones se atienden mediante el método GET. En este método, si queremos enviar información al servidor hemos de incluirla tras un carácter "?", seguido de un nombre de parámetro, seguido del carácter "=", seguido del valor. Los diferentes parámetros se separan mediante el carácter "&". Los espacios en blanco han de ser sustituidos por el carácter "+". En este ejemplo el parámetro *hl* corresponde al idioma de búsqueda y *q* a las palabras que hay que buscar.

3. Obtenemos la respuesta del servidor en una variable de tipo *string*.

4. Buscamos la primera aparición de "Aproximadamente" en la respuesta.
5. Tras esta palabra se encuentra la información que buscamos.

Obviamente, el correcto funcionamiento de esta aplicación está sujeto a que no se produzcan cambios en la forma en que Google visualiza los resultados. En caso de que se realice algún cambio en esta página, va a ser necesaria una adaptación de nuestra aplicación. La técnica de sacar información directamente de una página web se conoce como *web scraping*. Hoy en día existe otra alternativa más fiable para obtener información. En el siguiente apartado mostraremos cómo utilizar un servicio web con este propósito.



Ejercicio: *Búsquedas en Google con HTTP*

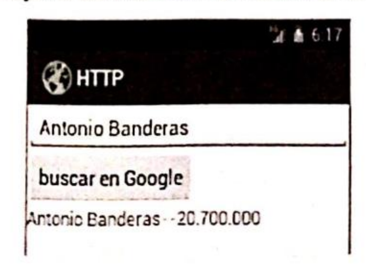
1. Crea un nuevo proyecto con nombre *HTTP* de tipo *Empty Activity.*
2. La aplicación debe solicitar el permiso de acceso a Internet, añadiendo en *AndroidManifest.xml* la siguiente línea:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

3. Reemplaza el código del *layout activity_main.xml* por:

```
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
   android:layout_width="match_parent"
   android:layout_height="match_parent">
   <EditText android:id="@+id/EditText01"
      android:layout_height="wrap_content"
      android:layout_width="match_parent"
      android:text="palabra a buscar"/>
   <Button android:id="@+id/Button01"
      android:layout_height="wrap_content"
      android:layout_width="wrap_content"
      android:onClick="buscar"
      android:text="buscar en Google"/>
   <TextView android:id="@+id/TextView01"
      android:layout_height="match_parent"
      android:layout_width="match_parent"
         android:textSize="8pt"/>
</LinearLayout>
```

La apariencia de este *layout* se muestra a continuación:



4. Reemplaza el código de *MainActivity.java* por:

```
public class MainActivity extends Activity {
   private EditText entrada;
   private TextView salida;

   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      entrada = findViewById(R.id.EditText01);
      salida = findViewById(R.id.TextView01);
      StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.
            Builder().permitNetwork().build());
   }

   public void buscar(View view){
      try {
         String palabras = entrada.getText().toString();
         String resultado = resultadosGoogle(palabras);
         salida.append(palabras + "--" + resultado + "\n");
      } catch (Exception e) {
         salida.append("Error al conectar\n");
         Log.e("HTTP", e.getMessage(), e);
      }
   }

   String resultadosGoogle(String palabras) throws Exception {
      String pagina = "", devuelve = "";
      URL url = new URL("http://www.google.es/search?hl=es&q=\""
            + URLEncoder.encode(palabras, "UTF-8") + "\"");
      HttpURLConnection conexion = url.openConnection();
      conexion.setRequestProperty("User-Agent",
            "Mozilla/5.0 (Windows NT 6.1)");
      if (conexion.getResponseCode()==HttpURLConnection.HTTP_OK) {
         BufferedReader reader = new BufferedReader(new
            InputStreamReader(conexion.getInputStream()));
         String linea = reader.readLine();
         while (linea != null) {
            pagina += linea;
            linea = reader.readLine();
         }
         reader.close();
         devuelve = buscaAproximadamente(pagina);
      } else {
         devuelve = "ERROR: " + conexion.getResponseMessage();
      }
      conexion.disconnect();
      return devuelve;
   }

   String buscaAproximadamente(String pagina){
      int ini = pagina.indexOf("Aproximadamente");
      if (ini != -1) {
```

```
            int fin = pagina.indexOf(" ", ini + 16);
            return pagina.substring(ini + 16, fin);
        } else {
            return "no encontrado";
        }
    }
}
```

El comienzo del código ha de resultarte familiar, posiblemente hasta la última línea del método `onCreate()`. En esta línea se configura StrictMode[45] para que permita accesos a la red desde el hilo principal.

Pasemos a describir el método `resultadosGoogle()`. Este método toma como entrada una secuencia de palabras y devuelve el número de veces que Google las ha encontrado en alguna página web. Lo primero que llama la atención es el modificador `throws Exception`. Estamos obligados a incluirlo si utilizamos la clase `HttpURLConnection`. Este modificador obliga a utilizar el método dentro de una sección `try ... catch ...` La razón de esto es que toda conexión HTTP es susceptible de no poder realizarse, por lo que tenemos la obligación de tomar las acciones pertinentes en caso de problemas.

Tras la declaración de variables, creamos la URL que utilizaremos para la conexión. El método `URLEncoder.encode()` se encargará de codificar las palabras en el formato esperado por el servidor. Entre otras cosas reemplaza espacios en blanco, caracteres no ASCII, etc. A continuación, preparamos la conexión por medio de la clase `HttpURLConnection`. Mediante el método `setRequestProperty()` podemos añadir cabeceras HTTP. En el punto 7 de este ejercicio se demuestra la necesidad de insertar la cabecera `User-Agent`.

En la siguiente línea se utiliza el método `getResponseCode()` para establecer la conexión. Si se establece sin problemas (`HTTP_OK`) leemos la respuesta línea a línea, concatenándola en el *string* `pagina`. El método `buscaAproximadamente()` busca en `pagina` la primera aparición de la palabra "Aproximadamente". Si la encontramos, buscamos el primer espacio a continuación del número que ha de aparecer tras "Aproximadamente" y devolvemos los caracteres entre ambas posiciones. En caso de no encontrar "Aproximadamente", puede que la secuencia buscada no se haya encontrado, aunque también es posible que Google haya cambiado la forma de devolver los resultados. En el caso de que la conexión no haya sido satisfactoria, devolvemos el mensaje de respuesta que nos dio el servidor `getResponseMessage()`.

5. Desde Android 9 el uso del protocolo HTTP no se recomienda, en su lugar hay que utilizar el protocolo HTTPS. La razón es que, en HTTP, toda la información transmitida entre cliente y servidor se envía sin encriptar. Esto lo hace muy vulnerable a fallos de seguridad. Si utilizamos este protocolo aparecerá un error en tiempo de ejecución y el LogCat se mostrará "Cleartext HTTP traffic to … not permitted". Toda la información a la que vamos a acceder en los siguientes ejercicios es pública, por lo tanto, no hay ningún problema en usar HTTP.

---

[45] http://developer.android.com/reference/android/os/StrictMode.html

Para permitir el tráfico HTTP en una aplicación Android debes añadir el siguiente atributo a la etiqueta <aplication> de AndroidManifest:

```
<application
   android:usesCleartextTraffic="true"
   ...
```

6. Ejecuta la aplicación y verifica que funciona correctamente.

7. En el código anterior comenta la línea:

```
conexion.setRequestProperty("User-Agent", ...);
```

8. Ejecuta el programa. Ahora el resultado ha de ser:

```
ERROR: Forbidden
```

En este caso, al tratar de establecer la conexión el servidor, en lugar de devolvernos el código de respuesta `200: OK`, nos ha devuelto el código `403: Forbidden`. ¿Por qué? La cabecera `User-Agent` informa al servidor de qué tipo de cliente ha establecido la conexión. Según se demuestra en este ejercicio, el servicio de búsquedas de Google prohíbe la respuesta a aquellos clientes que no se identifican.

9. Analiza el valor asignado a la cabecera "Mozilla/5.0 ...". Puedes comprobar que la información que estamos dando al servidor es totalmente errónea.

10. Modifica este valor por "Ejemplo de El gran libro de Android" y comprueba que el resultado es `403: Forbidden` ¿Por qué no quiere responder? La respuesta es que, desde finales de 2011, el servidor de Google exige que el tipo de navegador que se conecte sea conocido.

10.2.4. Uso de HTTP con AsyncTask

En el ejercicio anterior hemos configurado el modo estricto para que nos permita realizar accesos a la red desde el hilo principal. Aunque en la mayoría de los casos la interacción con el servidor es inferior a 2 décimas de segundo, podrían darse algunos casos en que la interacción fuera superior a un segundo (servidores sobrecargados o redes muy lentas). En estos casos, la interfaz de usuario permanecería bloqueada un tiempo excesivo.

En el capítulo 5 hemos aprendido a utilizar la clase `AsyncTask` para resolver estos problemas. El siguiente ejercicio nos muestra cómo puede utilizarse en este caso:

Ejercicio: *Búsquedas en Google con HTTP y AsyncTask*

1. Añade la siguiente clase dentro de `MainActivity` del ejercicio anterior:

```
class BuscarGoogle extends AsyncTask<String, Void, String> {
   private ProgressDialog progreso;

   @Override protected void onPreExecute() {
```

```
        progreso = new ProgressDialog(MainActivity.this);
        progreso.setProgressStyle(ProgressDialog.STYLE_SPINNER);
        progreso.setMessage("Accediendo a Google...");
        progreso.setCancelable(false); // false: no muestra botón cancelar
        progreso.show();
    }

    @Override protected String doInBackground(String... palabras) {
        try {
            return resultadosGoogle(palabras[0]);
        } catch (Exception e) {
            cancel(false); //true: interrumpimos hilo, false: dejamos termine
            Log.e("HTTP", e.getMessage(), e);
            return null;
        }
    }

    @Override protected void onPostExecute(String res) {
        progreso.dismiss();
        salida.append(res + "\n");
    }

    @Override protected void onCancelled() {
        progreso.dismiss();
        salida.append("Error al conectar\n");
    }
}
```

Observa como la nueva clase extiende `AsyncTask<String, Void, String>`. Se han escogido estos tres tipos de datos porque la entrada de la tarea será un `String` con las palabras que hay que buscar; no se necesita información de progreso y la salida será un `String` con el número de veces que se encuentran. En el método `onPreExecute()` se visualiza un `ProgressDialog` con estilo `SPINNER`. El método `doInBackground()` es el único que se ejecuta en un nuevo hilo. En él nos limitamos a llamar al método que habíamos programado en el ejercicio anterior para hacer la consulta. En caso de que se produzca una excepción cancelaremos la tarea, mostraremos el error en el `Log` y no devolveremos nada. El resto de los métodos se llaman según la tarea haya concluido o haya sido cancelada.

2. Añade el siguiente método:

```
public void buscar2(View view){
    String palabras = entrada.getText().toString();
    salida.append(palabras + "--");
    new BuscarGoogle().execute(palabras);
}
```

3. Abre el layout *activity_main.xml* y añade un botón con texto: "buscar en Google con AsyncTask" y con un valor para `onClick`: "buscar2".

4. Verifica que la aplicación funciona correctamente.



Preguntas de repaso: *El protocolo HTTP*

10.3. La librería Volley

Volley[46] es una librería que permite realizar peticiones HTTP de forma sencilla sin tener que preocuparnos de la gestión de hilos. No pertenece al API de Android, pero ha sido desarrollada por Google, por lo que es posible que sea incluida en un futuro. Presenta las siguientes ventajas:

Gestión automática de hilos: No tendrás que crear nuevos hilos o AsyncTasck de forma manual. Solo tendrás que escribir el escuchador adecuado cuando se produzca la descarga.

Caché transparente: Las descargas son guardadas de forma automática en disco o memoria. Si se solicita un contenido ya descargado, la respuesta será inmediata. La caché se maneja gracias a las cabeceras del protocolo HTTP (`Last-Modified`, `If-Modified-Since`...).

Manejo automático de colas de petición con prioridades: Volley ha sido diseñada para realizar múltiples descargar simultáneas, pero no se recomienda su uso para la descarga de grandes volúmenes de datos. En este caso es más interesante usar la clase `DownloadManager`.

10.3.1. Descargar un String con Volley

Para usar esta librería primero has de solicitar el permiso de Internet y añadir en Gradle la siguiente dependencia:

```
implementation 'com.android.volley:volley:1.1.1'
```

El siguiente paso es crear una cola de peticiones:

```
RequestQueue colaPeticiones = Volley.newRequestQueue(this);
```

Existen diferentes clases para crear peticiones. El siguiente nos devuelve el contenido en forma de `String`:

```
StringRequest peticion = new StringRequest(
      Request.Method.GET,
      "http://www.google.es/search?hl=es&q=busqueda",
      new Response.Listener<String>() {
         @Override
         public void onResponse(String respuesta) {
            ...
         }
      },
```

[46] https://developer.android.com/training/volley

```
        new Response.ErrorListener() {
           @Override
           public void onErrorResponse(VolleyError error) {
              ...
           }
        }
);
```

Tiene cuatro parámetros: el método a usar (GET, POST, HEAD, ...), la URL y dos escuchadores (uno para una respuesta satisfactoria y otro en caso de error).

Una vez creada la petición, la añadimos a la cola para que se ejecute:

```
colaPeticiones.add(peticion);
```

Existen cuatro clases según el tipo de datos a solicitar:

```
StringRequest(int method, String url, Response.Listener<String>
      listener, Response.ErrorListener errorListener)

ImageRequest(String url, Response.Listener<Bitmap> listener, int
      maxWidth, int maxHeight, Config decodeConfig,
      Response.ErrorListener errorListener)

JsonObjectRequest(int method, String url, JSONObject jsonRequest,
      Response.Listener<JSONObject> listener, Response.ErrorListener
      errorListener)

JsonArrayRequest(String url, Response.Listener<JSONArray> listener,
      Response.ErrorListener errorListener)
```

El parámetro `method` es optativo, de no indicarse se utiliza `GET`. Los constructores que no disponen de este parámetro utilizan por defecto `GET`. Las clases `JSONObject` y `JSONArray` pertenecen a la librería `org.json`, incluida en el API de Android.



Ejercicio: *Acceso HTTP con Volley*

1. Abre el proyecto HTTP desarrollado en el ejercicio "Utilizando HTTP desde Android", pero ahora utilizaremos la librería Volley en lugar de la clase `HttpURLConnection`.

2. Añade al fichero Gradle *Scripts/Bulid.gradle (Module:app)* la dependencia:

```
dependencies {
   ...
   implementation 'com.android.volley:volley:1.1.1'
}
```

3. Necesitamos crear una cola de peticiones. Añade en `MainActivity` la siguiente variable e inicializala en `onCreate()`:

```
private RequestQueue colaPeticiones;

@Override public void onCreate(Bundle savedInstanceState) {
```

```
   ...
   colaPeticiones = Volley.newRequestQueue(this);
}
```

No resulta recomendable crear una nueva cola cada vez que necesitemos hacer una petición, por eso, vamos a crear una única cola para toda la actividad. Si vas a usar Volley en toda la aplicación puede ser interesante declarar la cola en la clase `Application` o en un singleton para trabajar con una cola única (descrito en *El Gran Libro de Android Avanzado*).

4. En esta clase `MainActivity` añade el siguiente método:

```
void resultadosGoogleVolley(final String palabras) throws Exception {

   StringRequest peticion = new StringRequest(
         Request.Method.GET,
         "http://www.google.es/search?hl=es&q=\""
                     + URLEncoder.encode(palabras, "UTF-8") + "\"",
         new Response.Listener<String>() {
            @Override public void onResponse(String respuesta) {
               String resultado = buscaAproximadamente(respuesta);
               salida.append(palabras + "--" + resultado + "\n");
            }
         },
         new Response.ErrorListener() {
            @Override public void onErrorResponse(VolleyError error) {
               salida.append("Error: " + error.getMessage());
            }
         }
   ) {
        @Override
        public Map<String, String> getHeaders() throws AuthFailureError {
           Map<String, String> cabeceras = new HashMap<String, String>();
           cabeceras.put("User-Agent", "Mozilla/5.0 (Windows NT 6.1)");
           return cabeceras;
    }
   };
   colaPeticiones.add(peticion);
}
```

A diferencia de lo realizado en ejercicios anteriores, este método no nos devuelve un `String` con el resultado, por el contrario, va a modificar directamente la vista `salida`. Esto se debe a que Volley trabaja siempre de forma asíncrona. Los cuatro parámetros del constructor ya han sido explicados. Recuerda que el servidor de Google solo funcionaba si añadíamos una cabecera `User-Agent` válida. Para añadir esta cabecera, debemos sobrescribir el método `getHeaders()` de la clase.

También puedes sobrescribir `getMethod()` para cambiar el método o `getParams()` para añadir parámetros usando el método `POST`.

5. Añade el siguiente método:

```
public void buscar4(View view){
    String palabras = entrada.getText().toString();
    try {
        resultadosGoogleVolley(palabras);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

6. Abre el layout *activity_main.xml* y añade un botón con texto: "buscar en Google con Volley" y con un valor para `onClick`: "buscar4".

7. Verifica el funcionamiento de la aplicación.

10.3.2. Paso de parámetros con el método POST

Si quieres utilizar el método POST tentrás que indicarlo en el primer parámetro de la solicitud. Para las solicitudes `ImageRequest` y `JsonArrayRequest` no existe este parámetro y tendrás que configurarlo usando `getMethod()`.

Con el método POST los parámetros no se añaden a la URL, si no que son transmitidos tras las cabeceras. A continuación se muestra un ejemplo:

```
JsonArrayRequest peticion = new JsonArrayRequest(
        ...
) {
  @Override public Map<String,String> getParams() {
     Map<String,String> parametros = new HashMap<String,String>();
     parametros.put("hl", "es");
     parametros.put("q", "hola");
     return parametros;
  }
  @Override public int getMethod() { return Method.POST; }
};
```

10.3.3. Descargar imágenes con Volley

Disponemos de varias alternativas para descargar imágenes con Volley. La primera consiste en usar el método `ImageRequest()`, que trabaja de forma similar al mostrado en el apartado anterior:

```
ImageRequest peticion = new ImageRequest(
      "http://mmoviles.upv.es/img/moviles.png",
      new Response.Listener<Bitmap>() {
         @Override public void onResponse(Bitmap bitmap) {
            miImageView.setImageBitmap(bitmap);
         }
      }, 0, 0, null, // maxWidth, maxHeight, decodeConfig
      new Response.ErrorListener() {
         @Override public void onErrorResponse(VolleyError error) {
            miImageView.setImage Resource(R.drawable.error_carga);
         }
      }
```

```
);
colaPeticiones.add(peticion);
```

Observa como este método tiene tres parámetros adicionales, donde podemos configurar cómo se va a decodificar la imagen. Se utilizan los valores por defecto; para más información consultar la documentación oficial.

Cuando queremos descargar múltiples imágenes de forma simultánea se recomienda usar la clase `ImageLoader`. La principal diferencia con el método anterior es que las imágenes se guardan en una caché en memoria, en lugar de en disco. Esto agiliza mucho el proceso y evita molestos parpadeos de las imágenes.

El primer paso va a consistir en crear una instancia de `ImageLoader`:

```
RequestQueue colaPeticiones = Volley.newRequestQueue(this);
ImageLoader lectorImagenes = new ImageLoader(colaPeticiones,
   new ImageLoader.ImageCache() {
      private final LruCache<String, Bitmap> cache = new LruCache<String,
                                                         Bitmap>(10);
      public void putBitmap(String url, Bitmap bitmap) {
         cache.put(url, bitmap);
      }
      public Bitmap getBitmap(String url) {
        return cache.get(url);
      }
});
```

Como puedes ver un `ImageLoader` debe estar asociado a un `RequestQueue`. Además tiene que definir cómo se gestiona la caché por medio de un objeto `ImageCache`. En este objeto se define una estructura `LruCache` para almacenar en memoria pares de `URL-Bitmaps`. El valor 10 indica el máximo de elementos que queremos almacenar. Además, se definen dos métodos que permiten almacenar y recuperar elementos de la caché.

Resulta interesante declarar un solo `ImageLoader` y `RequestQueue` en toda la aplicación. Como hemos comentado en el apartado anterior, un buen sitio para hacerlo es en la clase `Application` o en un Singleton.

Usar el `ImageLoader` es muy sencillo. No tienes más que llamar al método `get()` e indicarle la URL y un escuchador:

```
lectorImagenes.get("http://mmoviles.upv.es/img/moviles.png",
       ImageLoader.getImageListener(miImageView, R.drawable.por_defecto,
                                             R.drawable.error_carga));
```

El escuchador será llamado cuando se descargue el `BitMap` y lo asignará al `ImageView` indicado. También se indican dos recursos que será asignados antes de la carga o en caso de error.

Disponemos de una tercera alternativa que consiste en usar la vista `NetworkImageView`, definida en Volley para trabajar conjuntamente con un `ImageLoader`. La nueva vista reemplazaría a `ImageView`, pero incorpora la posibilidad de cargar la imagen desde una URL. Trabajar con esta vista tiene la ventaja de que

la descarga se puede sincronizar con la visualización: cuando la vista va a verse se puede iniciar la descarga y cuando deja de verse se puede cancelar la descarga.

Para usar esta alternativa reemplaza en un layout la etiqueta `ImageView` por la siguiente, dejando los atributos igual:

```
<com.android.volley.toolbox.NetworkImageView
    android:id="@+id/icono"
    android:layout_width=" match_parent"
    android:layout_height="match_parent"
    …"/>
```

Para asociar la URL utiliza el siguiente código:

```
icono = itemView.findViewById(R.id.icono);
icono.setImageUrl("http://mmoviles.upv.es/img/moviles.png", lectorImagenes);
```



Ejercicio: *Cargar imágenes de un RecyclerView con Volley*

Cuando se trabaja con una lista generada con `RecyclerView` es muy frecuente que cada elemento contenga una imagen que ha de descargarse de una URL. Se van a realizar múltiples peticiones simultáneas, por lo que en este caso, se recomienda el uso de `ImageLoader`.

1. Abre el proyecto Asteroides y añade al fichero Gradle *Scripts/Bulid.gradle (Module:app)* la dependencia: `'com.android.volley:volley:1.1.1'`.
2. En la clase `MainActivity` declara las variables:

```
public static RequestQueue colaPeticiones;
public static ImageLoader lectorImagenes;
```

3. En el método `onCreate()` inicializa estas variables como se acaba de ver. Recuerda que ya están declarados globalmente y debes quitar la clase antes del nombre del objeto.
4. En la clase `MiAdaptador` dentro de `onBindViewHolder()` comenta el código:

```
switch (Math.round((float)Math.random()*3)){
    case 0:
        holder.icon.setImageResource(R.drawable.asteroide1);
        break;
    …
}
```

y reemplázalo por:

```
MainActivity.lectorImagenes.get("http://mmoviles.upv.es/img/moviles.png",
        ImageLoader.getImageListener(holder.icon, R.drawable.asteroide1,
                                                  R.drawable.asteroide3));
```

5. Verifica el funcionamiento.



Ejercicio: *Cargar imágenes de un RecyclerView con NetworkImageView*

En el ejercicio anterior hemos trabajado con vistas `ImageView`. En este, vamos a reemplazarlas por `NetworkImageView`. De esta forma la gestión de la descarga puede sincronizarse con la visualización, obteniendo unos resultados óptimos.

1. Edita el layout *elemento_lista.xml* reemplazando el `<ImageView…` por `<com.android.volley.toolbox.NetworkImageView….`
2. En la clase `MiAdaptador`, dentro de la clase `ViewHolder`, reemplaza las dos apariciones de `ImageView` por `NetworkImageView`.
3. En la clase `MiAdaptador`, dentro de `onBindViewHolder()`, comenta el código introducido en el apartado anterior y reemplázalo por:

```
holder.icon.setImageUrl("http://mmoviles.upv.es/img/moviles.png",
                        MainActivity.lectorImagenes);
```

4. Verifica el resultado.



Preguntas de repaso: *La librería Volley*

10.4. Servicios web

La W3C define "servicio web" como un sistema de *software* diseñado para permitir interoperabilidad máquina a máquina en una red. Se trata de API que son publicadas, localizadas e invocadas a través de la web. Es decir, una vez desarrolladas, son instaladas en un servidor, y otras aplicaciones (u otros servicios web) pueden descubrirlas desde otros ordenadores de Internet e invocar uno de sus servicios.

Como norma general, el transporte de los datos se realiza a través del protocolo HTTP y la representación de los datos mediante XML. Sin embargo, no hay reglas fijas en los servicios web y en la práctica no tiene por qué ser así.

Una de las grandes ventajas de este planteamiento es que es tecnológicamente neutral. Es decir, podemos utilizar un servicio web sin importarnos el sistema operativo o el lenguaje en el que fue programado. Además, al apoyarse sobre el protocolo HTTP, puede utilizar los sistemas de seguridad (*https*) y presenta pocos problemas con cortafuegos, al utilizar puertos que suelen estar abiertos (80 o 8080).

Como inconveniente podemos resaltar que, dado que el intercambio de datos se realiza en formato de texto (XML), tiene menor rendimiento que otras alternativas como RMI (*Remote Method Invocation*), CORBA (*Common Object Request Broker Architecture*) o DCOM (*Distributed Component Object Model*). Además, el hecho de apoyarse en HTTP hace que resulte complicado para un cortafuego filtrar este tipo

de tráfico. ¿No acabamos de decir que esto era una ventaja? Es posible que lo que para un desarrollador sea una ventaja, para un administrador de red sea un inconveniente.

10.4.1. Alternativas en los servicios web

Como acabamos de ver, el término "servicio web" resulta difícil de definir de forma precisa. En torno a este concepto se han desarrollado varias propuestas bastante diferentes entre sí. En este apartado estudiaremos las dos alternativas que están teniendo más relevancia en la actualidad: SOAP y REST. No obstante, dada la complejidad que surge de estas propuestas, resulta interesante centrar algunos conceptos antes de empezar a describir estas alternativas. Comenzaremos indicando que existen tres enfoques diferentes a la hora de definir un servicio web. Es lo que se conoce como arquitectura del servicio web.

Llamadas a procedimiento remotos (RPC): Se enfoca el servicio web como una colección de operaciones o procedimientos que pueden ser invocados desde una máquina diferente de donde se ejecutan. Como RPC es una extensión directa del paradigma de llamadas a funciones, resulta sencillo de entender para un programador. Al ser una de las primeras alternativas que se implementó, se conocen como servicios web de primera generación.

Arquitectura orientada a servicios (SOAP): En el planteamiento anterior, RPC, la unidad básica de interacción es la operación. En este nuevo planteamiento, la unidad de interacción pasa a ser el mensaje. Por lo tanto, en muchos casos se conocen como servicios orientados a mensaje. Cada uno de los mensajes que vamos a utilizar ha de ser definido siguiendo una estricta sintaxis expresada en XML. En la actualidad se trata de la arquitectura más extendida, soportada por la mayoría del *software* de servicios web.

Transferencia de estado representacional (REST): En los últimos años se está popularizando este nuevo planteamiento, que se caracteriza principalmente por su simplicidad. En REST se utiliza directamente el protocolo HTTP, por medio de sus operaciones GET, POST, PUT y DELETE. En consecuencia, esta arquitectura se centra en la solicitud de recursos, en lugar de las operaciones o los mensajes de las alternativas anteriores.

Servicios web basados en SOAP

SOAP (*Simple Object Access Protocol*) es el protocolo más utilizado en la actualidad para implementar servicios web. Fue creado por Microsoft, IBM y otros, aunque en la actualidad está bajo el auspicio de la W3C.

Utiliza como transporte HTTP, aunque también es posible utilizar otros métodos de transporte, como el correo electrónico. Los mensajes del protocolo se definen utilizando un estricto formato XML, que ha de ser consensuado por ambas partes. A continuación se muestra un posible ejemplo de mensaje SOAP:

```
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing">
    <soapenv:Header>
```

```
        <wsa:MessageID>
            uuid:920C5190-0B8F-11D9-8CED-F22EDEEBF7E5
        </wsa:MessageID>
        <wsa:To>
            http://localhost:8081/axis/services/BankPort
        </wsa:To>
    </soapenv:Header>
    <soapenv:Body>
        <axis2:echo xmlns:axis2="http://ws.apache.org/axis2">
            Hello World
        </axis2:echo>
    </soapenv:Body>
</soapenv:Envelope>
```

Un mensaje SOAP contiene una etiqueta `<Envelope>`, que encapsula las etiquetas `<Header>` y `<Body>`. La etiqueta `<Header>` es opcional y encapsula aspectos relativos a calidad del servicio, como seguridad, esquemas de direccionamiento, etc. La cabecera `<Body>` es obligatoria y contiene la información que las aplicaciones quieren intercambiar.

SOAP proporciona una descripción completa de las operaciones que puede realizar un nodo mediante una descripción WSDL (*Web Services Description Language*), por supuesto codificada en XML. En uno de los siguientes apartados crearemos un servicio web y podremos estudiar el fichero WSDL correspondiente.

Aunque SOAP está ampliamente extendido como estándar para el desarrollo de servicios web, no resulta muy adecuado para ser utilizado en Android. Esto es debido a la complejidad introducida, supone una sobrecarga que implica un menor rendimiento frente a otras alternativas como REST. Además, Android no incorpora las librerías necesarias para trabajar con SOAP.

No obstante, es posible que ya dispongas de un servidor basado en SOAP y necesites implementar un cliente en Android. En tal caso puedes utilizar las librerías kSOAP2 (http://ksoap2.sourceforge.net/).

Servicios web basados en REST

En primer lugar conviene destacar que el término REST se refiere a una arquitectura en lugar de a un protocolo en concreto, como es el caso de SOAP. A diferencia de SOAP, no vamos a añadir una capa adicional a la pila de protocolos, si no que utilizaremos directamente el protocolo HTTP. Siendo estrictos, la arquitectura REST no impone el uso de HTTP; no obstante, en la práctica se entiende que un servicio web basado en REST es aquel que se implementa directamente sobre la web.

Este planteamiento supone seguir los principios de la aplicación WWW, pero en lugar de solicitar páginas web solicitaremos servicios web. Los principios básicos de la aplicación WWW y, por tanto, los de REST son:

- Transporte de datos mediante HTTP, utilizando las operaciones de este protocolo, que son GET, POST, PUT y DELETE.
- Los diferentes servicios son invocados mediante el espacio de URI unificado. Como ya se ha tratado en este libro, una URI identifica un recurso

en Internet. Este sistema ha demostrado ser flexible, sencillo y potente al mismo tiempo. Se cree que fue uno de los principales factores que motivó el éxito de WWW.

- La codificación de datos es identificada mediante tipos MIME (*text/html*, *image/gif*, etc.), aunque el tipo de codificación preferido es XML (*text/xml*).

Las ventajas de REST derivan de su simplicidad. Entre estas podemos destacar: mejores tiempos de respuesta y disminución de sobrecarga tanto en cliente como en servidor, mayor estabilidad frente a futuros cambios y, también, una gran sencillez en el desarrollo de clientes, que solo han de ser capaces de realizar interacciones HTTP y codificar información en XML.

Como inconveniente podemos indicar que, al igual que ocurre con el protocolo HTTP, no se mantiene el estado. Es decir, cada solicitud es tratada por el servidor de forma independiente sin recordar solicitudes anteriores.



Ejercicio: *Comparativa entre una interacción SOAP y REST*

NOTA: *El servidor ya no está operativo.*

La empresa WebserviceX.NET ofrece un servicio web, GetIPService, que nos permite conocer, a partir de una dirección IP, el país al que pertenece. Este servicio puede ser utilizado tanto con REST como con SOAP, lo cual nos va a permitir comparar ambos mecanismos. Más todavía, dentro de REST tenemos dos opciones para mandar datos al servidor: el método GET y el método POST. El servicio que vamos a probar nos permite las dos variantes, lo que nos permitirá comparar ambos mecanismos.

1. Abre un navegador web y accede a la URL:

   http://www.webservicex.net/geoipservice.asmx/GetGeoIP?IPAddress=158.42.38.1

2. Verifica que el resultado es similar al siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<GeoIP xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns="http://www.webservicex.net/">
   <ReturnCode>1</ReturnCode>
   <IP>158.42.38.1</IP>
   <ReturnCodeDetails>Success</ReturnCodeDetails>
   <CountryName>European Union</CountryName>
   <CountryCode>EU</CountryCode>
</GeoIP>
```

3. Prueba otras IP al azar y verifica a qué países pertenecen.

4. Vamos a emular el protocolo HTTP de forma similar a como lo hemos hecho en apartados anteriores. Desde un intérprete de comandos (símbolo del sistema/*shell*) escribe:

```
telnet www.webservicex.net 80
```

5. Cuando se establezca la conexión teclea exactamente el siguiente código seguido de un salto de línea adicional (↵):

```
GET /geoipservice.asmx/GetGeoIP?IPAddress=158.42.38.1 HTTP/1.1
Host: www.webservicex.net
```

NOTA: También puedes cortar el texto y pegarlo. Para pegar sobre la ventana de símbolo de sistema de Windows has de pulsar con el botón derecho del ratón y seleccionar Pegar.

6. El resultado ha de parecerse al anterior, aunque al principio el servidor enviará sus cabeceras:

```
HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Content-Length: 374
Content-Type: text/xml; charset=utf-8
Server: Microsoft-IIS/7.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Mon, 30 Jan 2012 19:28:55 GMT
```

7. Como acabamos de ver, el protocolo HTTP permite enviar información al servidor utilizando el método GET e introduciendo un carácter "?" al final de la URL seguido de los parámetros. El protocolo HTTP también permite mandar información con el método POST. El servicio web que estamos utilizando nos permite las dos alternativas. Veamos en qué consiste el método POST. Escribe en el intérprete de comandos:

```
telnet www.webservicex.net 80
```

8. Cuando se establezca la conexión pega los siguientes caracteres:

```
POST /geoipservice.asmx/GetGeoIP HTTP/1.1
Host: www.webservicex.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 21

IPAddress=158.42.38.1
```

Como puedes observar, la información enviada es la misma, aunque ahora en lugar de adjuntarla a la URL se manda tras las cabeceras separada por una línea en blanco.

NOTA: La cabecera `Content-Length:` es obligatoria. Indica el número de caracteres enviados. En caso de que cambiara la longitud de la dirección IP tendrías que ajustarlo.

9. El servidor está esperando nuevos comandos, la conexión todavía está abierta. Pulsa *Ctrl-C* para cerrar la conexión.

10. Ahora vamos a usar el mismo servicio, aunque mediante el protocolo SOAP 1.1 (*NOTA: también es posible utilizar SOAP 1.2*). Escribe en el intérprete de comandos:

```
telnet www.webservicex.net 80
```

11. Cuando se establezca la conexión pega los siguientes caracteres:

```
POST /geoipservice.asmx HTTP/1.1
Host: www.webservicex.net
Content-Type: text/xml; charset=utf-8
Content-Length: 371
SOAPAction: "http://www.webservicex.net/GetGeoIP"
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <GetGeoIP xmlns="http://www.webservicex.net/">
      <IPAddress>158.42.38.1</IPAddress>
    </GetGeoIP>
  </soap:Body>
</soap:Envelope>
```

12. Pulsa *Ctrl-C* para cerrar la conexión. Compara la información mandada en SOAP con el caso anterior. El resultado obtenido ha de ser similar al siguiente:

```
HTTP/1.1 200 OK
Cache-Control: private, max-age=0
Content-Length: 514
Content-Type: text/xml; charset=utf-8
Server: Microsoft-IIS/7.0
X-AspNet-Version: 4.0.30319
X-Powered-By: ASP.NET
Date: Mon, 30 Jan 2012 20:07:55 GMT

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <soap:Body>
    <GetGeoIPResponse xmlns="http://www.webservicex.net/">
      <GetGeoIPResult>
        <ReturnCode>1</ReturnCode>
        <IP>158.42.38.1</IP>
        <ReturnCodeDetails>Success</ReturnCodeDetails>
        <CountryName>European Union</CountryName>
        <CountryCode>EU</CountryCode>
      </GetGeoIPResult>
    </GetGeoIPResponse>
  </soap:Body>
</soap:Envelope>
```



Recursos adicionales: *Ejemplos de algunos servicios web gratuitos*

En la siguiente tabla te mostramos una lista con algunos servicios web:

Nombre	Descripción	Empresa	Tipo (codific.)
Google Custom Search	Búsqueda en Web con respuesta JSON o Atom. https://www.googleapis.com/customsearch/v1?key=KEY&cx=01757666251246:omuauf_lfve&q=Antonio+Banderas	Google	REST (JSON/ XML)
Books API	Búsqueda y altas de libros. No hay que registrar clave. Obsoleto (aunque sigue funcionando). http://books.google.com/books/feeds/volumes?q=nadal	Google	REST (XML)
Books API (nueva)	Búsqueda y altas de libros. https://www.googleapis.com/books/v1/volumes?q=nadal	Google	REST (JSON)
World Digital Library	Biblioteca Digital Mundial http://api.wdl.org/	Unesco	REST (XML...)
Google Maps Geocoding	A partir de una dirección nos da la longitud y latitud. O a la inversa. http://maps.google.com/maps/api/geocode/xml?address=Gandia	Google	REST (JSON, XML)
Foreign exchange	Cambio entre divisas: http://data.fixer.io/api/latest?base=EUR&symbols=USD&access_key=083861d3551dcb9eab0ee576a30c420d	Fixer	REST (JSON)
Eurofxref	Cambio del Euro: https://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml	Banco Central Europeo	REST (XML)
CIMA	Información medicamentos: https://cima.aemps.es/cima/rest/medicamento?nregistro=51347	Agencia Española Medicam	REST (JSON)
Valencia Datos Abiertos[47]	Monumentos, fallas, contaminación, trafico, aparcamiento, autobus, recursos sociales, urbanismo, etc. http://mapas.valencia.es/lanzadera/opendata/TRA_CARGAYDESCARGA/JSON	Ayuntamiento de Valencia	REST (JSON, KML, ...)
Datos abiertos en España	Portal que recopila datos en abierto de administraciones españolas: ministerios, comunidades... http://datos.gob.es/	Gobierno de España	SOAP /REST (XML, ...)

[47] http://gobiernoabierto.valencia.es/

Otro sitio interesante es el que ofrece el Ministerio de Fomento de España[48]. Aquí encontrarás un directorio con centenares de servicios web geográficos ofrecidos por instituciones públicas (por ejemplo, el precio del combustible en estaciones de servicio).

10.4.2. Acceso a servicios web de terceros

En este apartado nos centraremos en cómo utilizar un servicio REST publicado por un tercero. Como se ha comentado, el acceso a un servicio SOAP resulta algo más complicado y Android no dispone de librerías para facilitarnos el trabajo.



Ejercicio: *Acceso a servicios web de búsqueda de libros*

En concreto vamos a utilizar el servicio Books API, que permite buscar y gestionar libros de la base de datos de Google. Es un servicio obsoleto, aunque actualmente operativo. También se puede utilizar el nuevo servicio que ofrece Google (nueva Books API), que nos da el resultado en JSON en lugar de en XML.

1. Para probar el servicio, abre un navegador web y accede a la siguiente URL:

   http://books.google.com/books/feeds/volumes?q=antonio+banderas

   El resultado ha de ser similar a:

```
<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns:openSearch="http://a9.com/-/spec/opensearchrss/1.0/"
      xmlns:gbs=http://schemas.google.com/books/2008
      xmlns:dc=http://purl.org/dc/terms
      xmlns:batch=http://schemas.google.com/gdata/batch
      xmlns:gd=http://schemas.google.com/g/2005
      xmlns="http://www.w3.org/2005/Atom" >
  <id >http://www.google.com/books/feeds/volumes</id>
  <updated >2012-01-30T23:56:34.000Z</updated>
  <category scheme=http://schemas.google.com/g/2005#kind
     term="http://schemas.google.com/books/2008#volume" />
   <title type="text" > Search results for antonio banderas</title>
  <link href="http://www.google.com" rel="alternate" type="text/html"/>
  <link href=http://www.google.com/books/feeds/volumes
     rel="http://schemas.google.com/g/2005#feed"
     type="application/atom+xml" />
  <link
     href=http://www.google.com/books/feeds/volumes?q=antonio+banderas
     rel="self" type="application/atom+xml" />
  <link href="http://www.google.com/books/feeds/volumes?q=antonio
     +banderas&amp; start-index=11&amp;max-results=10" rel="next"
     type="application/atom+xml" />
```

48 http://www.idee.es/web/guest/directorio-de-servicios

```
<author><name>Google Books Search</name>
   <uri>http://www.google.com</uri></author>
<generator version="beta" >Google Book Search data API</generator>
<openSearch:totalResults >596</openSearch:totalResults>
<openSearch:startIndex >1</openSearch:startIndex>
<openSearch:itemsPerPage >10</openSearch:itemsPerPage>
<entry >
   <id >http://www.google.com/books/feeds/volumes/_YS1OAAACAAJ</id>
   <updated >2012-01-30T23:56:34.000Z</updated>
...
```

En el resultado obtenido, localiza la etiqueta `<totalResults>`. Representa los libros encontrados que contienen la búsqueda "antonio banderas".

2. Abre el proyecto *HTTP* creado en el ejercicio *"Utilizando HTTP desde Android"*. Ahora utilizaremos un servicio web de búsqueda de libros, en lugar de buscar la información en una página web.

3. En la actividad principal añade el siguiente método:

```
String resultadosSW(String palabras) throws Exception {
   URL url = new URL("http://books.google.com/books/feeds/volumes?q=\""
         + URLEncoder.encode(palabras, "UTF-8") + "\"");
   SAXParserFactory fabrica = SAXParserFactory.newInstance();
   SAXParser parser = fabrica.newSAXParser();
   XMLReader lector = parser.getXMLReader();
   ManejadorXML manejadorXML = new ManejadorXML();
   lector.setContentHandler(manejadorXML);
   lector.parse(new InputSource(url.openStream()));
   return manejadorXML.getTotalResults();
}
```

El método comienza creando la URL de acceso. El resto del código utiliza las librerías `org.xml.sax.*` para realizar un proceso de *parser* sobre el código XML de la URL y así extraer la información que nos interesa. Este proceso se ha explicado en el capítulo anterior. El trabajo que sí que tendremos que realizar en función del formato XML específico será la creación de la clase `XMLHandler`. Una vez finalizado el *parser*, podemos llamar al método `getTotalResults()` de nuestro manejador para que nos devuelva la información que nos interesa.

4. Realiza una copia del método `buscar()` cambiando el nombre por `buscar3()`. En este método reemplaza `resultadosGoogle()` por `resultadosSW()`.

5. Abre el layout *activity_main.xml* y añade un botón con texto: "buscar en SW libros" y con un valor para `onClick`: "buscar3".

6. A continuación mostramos la definición de la clase `ManejadorXML`. Copia este código dentro de la clase `MainActivity`:

```
public class ManejadorXML extends DefaultHandler {

   private String totalResults;
   private StringBuilder cadena = new StringBuilder();

   public String getTotalResults() {
```

```
      return totalResults;
   }

   @Override
   public void startElement(String uri, String nombreLocal, String
         nombreCualif, Attributes atributos) throws SAXException {
      cadena.setLength(0);
   }

   @Override
   public void characters(char ch[], int comienzo, int lon){
      cadena.append(ch, comienzo, lon);
   }

   @Override
   public void endElement(String uri, String nombreLocal,
      String nombreCualif) throws SAXException {
      if (nombreLocal.equals("totalResults")) {
         totalResults = cadena.toString();
      }
   }
}
```

Para procesar el fichero XML extendemos la clase `DefaultHandler` y rescribimos muchos de sus métodos: en `startElement()` inicializamos la variable `cadena` cada vez que empieza una etiqueta; en el método `characters()` añadimos el contenido que aparece dentro de la etiqueta; finalmente, en `endElement()` recogemos el valor acumulado en `cadena` cada vez que termina una etiqueta. Como hemos comentado, de todo el código XML que vamos a procesar, solo nos interesa el contenido de la etiqueta `<totalResults>`.

7. Verifica el funcionamiento de la aplicación.



Desafío: *Convertidor de divisas mediante un servicio web*

Un servicio Web, ofrecidos por Fixer, permite obtener la ratio de conversión de divisas según el cambio actual. A continuación, se muestra un ejemplo de uso para obtener la ratio euro-dólar:

http://data.fixer.io/api/latest?base=EUR&symbols=USD&access_key=083861d3551dcb9eab0ee576a30c420d

1. Entra en https://fixer.io y crea un nuevo usuario. Solicita un `access_key` en el plan gratuito. Reemplaza el código obtenido en la URL anterior y verifica que funciona correctamente.
2. Retoma el diseño de *layouts* de la eurocalculadora, realizado en el capítulo 2, para un nuevo proyecto de conversión de divisas. En una primera fase solo se realizará la conversión de euros a dólares, aplicando la ratio obtenida a través del servicio web indicado.

3. En una segunda fase puedes permitir que el usuario seleccione la divisa de entrada (reemplazando en la URL, "EUR") y la de salida (reemplazando en la URL, "USD"). Puedes encontrar una lista de las divisas disponibles en la URL http://data.fixer.io/api/latest?base=EUR&access_key=083861d3551dcb9eab0ee576a30c420d.

10.4.3. Un servicio web con Apache, PHP y MySQL

A la hora de desarrollar una aplicación distribuida, una de las alternativas más utilizadas en la actualidad son los servicios web. A lo largo de este apartado y el siguiente aprenderás cómo crear tus propios servicios web e instalarlos en un servidor web. Como no podría ser de otra manera, el ejemplo seleccionado es el mismo que ya hemos implementado en varias ocasiones: un almacén de las mejores puntuaciones de Asteroides.

No resultaría demasiado complicado rescribir el ejemplo del servidor de ECHO descrito en un apartado anterior para crear nuestro propio servidor web. No obstante, si estamos trabajando en un entorno empresarial, esta alternativa no sería la más adecuada. En este entorno es mucho más recomendable utilizar un servidor web de uso comercial, como Apache. Esta solución resulta mucho más segura y escalable.

En el siguiente apartado aprenderemos a crear un servicio web usando la combinación Apache, Tomcat y Axis2. Esta opción tiene varias ventajas: toda la programación la hacemos con un mismo lenguaje (Java), el código que tenemos que escribir es muy limpio y permite publicar el servicio con estilo REST o SOAP.

En este apartado estudiaremos otra alternativa, que consiste en usar el trinomio Apache, PHP y MySQL. Tiene sus inconvenientes, como la necesidad de usar un nuevo lenguaje (PHP), y el código a usar es más rudimentario. Sin embargo, presenta importantes ventajas: se trata de la solución más extendida. La mayoría de las empresas ya disponen de un servidor web basado en Apache, PHP y MySQL. Siempre será conveniente montar el servicio web usando la misma tecnología que la que usamos en el sitio web. Por otra parte, la mayoría de los servidores de *hosting* comerciales trabajan con PHP y MySQL.



Ejercicio: *Instalación de Apache, PHP y MySQL*

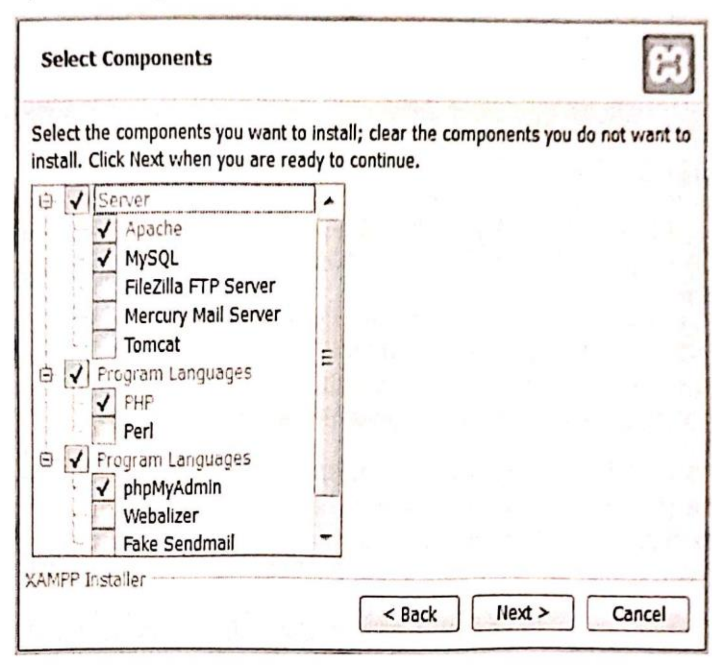
En este ejercicio vamos a instalar en nuestro propio ordenador el servidor web Apache con su extensión para poder ejecutar código PHP. Además del servidor de bases de datos MySQL. Este proceso puede realizarse de forma muy sencilla y rápida utilizando el paquete de *software* XAMPP. Además incorpora algunas herramientas para poder administrar estos servidores muy fácilmente. No obstante, usar un servidor de *hosting* comercial puede ser una alternativa más sencilla y segura. Si no estás interesado en instalar tu propio servidor web, puedes saltarte este ejercicio y realizar el ejercicio que encontrarás más adelante.

1. Descarga la última versión de XAMPP de la siguiente web:

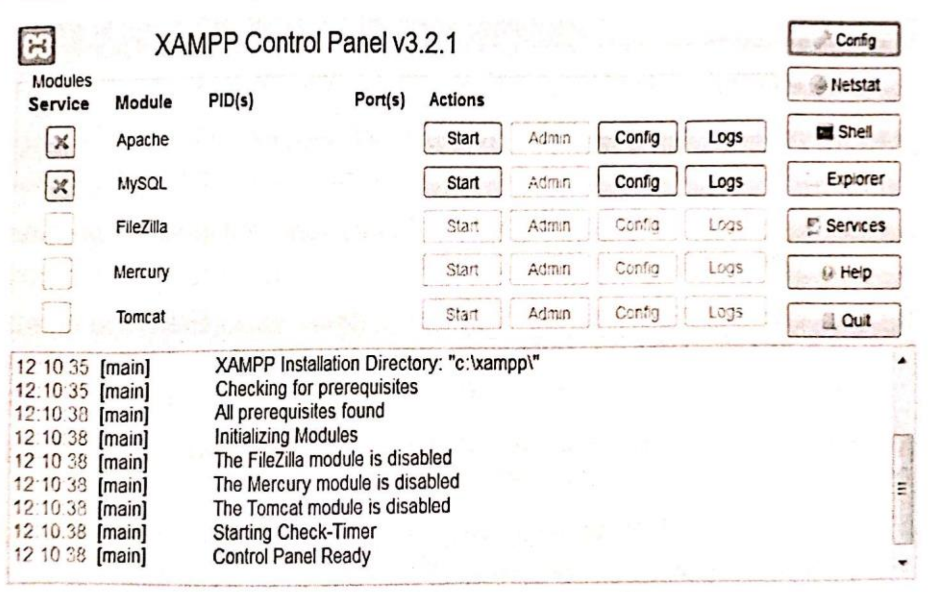
http://www.apachefriends.org/en/xampp.html

Encontrarás versiones para Linux, Windows y Mac. En este ejercicio hemos instalado la versión 1.8.2 para Windows usando el ejecutable.

2. Inicia el proceso de instalación según tu SO. Los componentes mínimos necesarios para este ejercicio se indican a continuación:



3. Una vez instalado ejecuta *XAMPP Control Panel*:



4. Pulsa los botones *Start*, tanto para Apache como para MySQL.

5. Para verificar que el servidor web está en marcha, abre un navegador y desde la barra de direcciones accede a http://localhost. Se mostrará una página con información sobre XAMPP.



Ejercicio: *Configuración de Apache*

En este ejercicio veremos una visión superficial sobre la configuración del servidor web Apache.

1. Verifica que el servidor está arrancado accediendo a la dirección http://localhost. Se utiliza para referirte a tu propia dirección IP. Es equivalente a escribir http://127.0.0.1.
2. Dentro de la carpeta donde hayas instalado XAMPP (por ejemplo, *C:/xampp*) abre la carpeta *htdocs*. Dentro encontrarás todos los ficheros que publica el servidor.
3. El fichero que toma por defecto en esta carpeta es *index.php* (o *index.html* si no lo encuentra). Edita este fichero y estudia su estructura. Modifica este fichero y recarga en el navegador para observar los cambios (http://localhost).
4. Ejecuta *XAMPP Control Panel* y pulsa el botón *Config* de Apache. Selecciona Apache (*httpd.conf*). Se editará el fichero *xampp/apache/conf/httpd.conf*. Es un fichero bastante largo, aunque normalmente solo tendrás que modificar los siguientes parámetros:
   - `Listen 80` - Indica que el servidor escucha el puerto 80. Es frecuente cambiar este valor por `8080` o `888`. Para aceptar conexiones solo de este *host*, cambia la línea por `Listen 127.0.0.1:80`.
   - `ServerAdmin postmaster@localhost` - Correo electrónico del administrador del servidor.
   - `ServerName localhost:80` - Nombre del servidor, indicando nombre de dominio y puerto. Si no se indica, trata de obtenerlo automáticamente.
   - `DocumentRoot "C:/xampp/htdocs"` - El directorio donde se encuentran los documentos servidos por el servidor.
   - `DirectoryIndex index.php index.pl … index.html` - Establece el archivo que Apache ofrece cuando se solicita un directorio sin indicar un archivo concreto. De encontrar varios se escoge el que esté antes en esta lista.

   Si modificas algún valor, recuerda guardar el fichero y reinicializar Apache para que se carguen los nuevos valores.
5. Vamos a probar si el servidor web es accesible desde otros dispositivos conectados a tu red de área local. Utiliza el comando `ipconfig` (Windows) o `ifconfig` (Linux/Mac) para averiguar la dirección IP de tu ordenador.
6. Abre un navegador desde otro dispositivo y accede a la dirección que acabas de obtener. Si lo haces desde un móvil, has de acceder a través de Wi-Fi. Si tienes problemas, es posible que sea culpa del cortafuegos. En este caso tendrás que desactivarlo.



Ejercicio: *Un servicio web con PHP y MySQL*

En este ejercicio comenzamos creando una base de datos y luego escribiremos un par de ficheros PHP que implementarán las dos acciones del servicio web *puntuaciones*.

1. Ejecuta *XAMPP Control Panel* y asegúrate de que tanto Apache como MySQL están arrancados.

2. Pulsa el botón *Admin* de MySQL que encontrarás en *XAMPP Control Panel*. De esta forma accedemos a la herramienta de administración del servidor de bases de datos phpMyAdmin.

3. Selecciona la lengüeta *SQL* e introduce las siguientes instrucciones en el cuadro de texto:

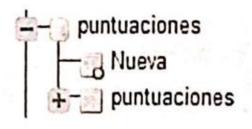
```
CREATE DATABASE IF NOT EXISTS puntuaciones;
USE puntuaciones;

CREATE TABLE puntuaciones (
  _id INTEGER PRIMARY KEY AUTO_INCREMENT,
  puntos INTEGER, nombre TEXT, fecha BIGINT);

INSERT INTO puntuaciones (puntos, nombre, fecha) VALUES
  (10000, 'Pedro', 0),
  (20000, 'Rosa', 0);
```

4. Pulsa el botón *OK* para ejecutar estas sentencias.

5. En el marco de la izquierda, pulsa el botón verde con forma de recargar. Observa como en la lista de bases de datos aparece *puntuaciones*. Si pulsas en el botón + de su izquierda se mostrarán sus tablas.



6. Selecciona la tabla *puntuaciones* para examinar su contenido.

←T→			▾	_id	puntos	nombre	fecha
☐	Editar	Copiar	Borrar	1	10000	Pedro	0
☐	Editar	Copiar	Borrar	2	20000	Rosa	0

Como puedes observar, disponemos de diferentes herramientas para editar los valores de la tabla. Por ejemplo, podemos utilizar la lengüeta *Insertar* para añadir una nueva fila a la tabla.

7. Explora otras utilidades que nos ofrece phpMyAdmin para trabajar con bases de datos.

8. Dentro de la carpeta donde hayas instalado XAMPP (por ejemplo, *C:/xampp*), abre la carpeta *htdocs*. Crea dentro la carpeta *puntuaciones*.

9. Dentro de esta carpeta crea el fichero *lista.php* con el siguiente contenido:

```
<?php
  $con = new mysqli('localhost', 'root', '', 'puntuaciones');
  if ($con->connect_errno) {
     echo 'Error al conectar base de datos: ', $con->connect_error;
     exit();
```

```
   }
   $sql = 'SELECT puntos, nombre FROM puntuaciones ORDER BY fecha DESC';
   if (isset($_GET['max'])) {
      $sql .= ' LIMIT ?';
   }
   $cursor = $con->stmt_init();
   if ($cursor->prepare($sql)) {
      if (isset($_GET['max'])) {
         $cursor->bind_param("s",$_GET['max']);
      }
      $cursor->execute();
      $cursor->bind_result($puntos, $nombre);
      while($cursor->fetch()) {
         echo $puntos.' '.$nombre."\n";
      }
      echo "\n";
      $cursor->close();
   }
   $con->close();
?>
```

El código PHP suele estar entremezclado entre el código HTML. Para diferenciarlo de este, hay que introducirlo entre los caracteres `<?php` y `?>`. La primera sentencia establece una conexión a una base de datos situada en un servidor MySQL. Necesita cuatro parámetros: primero, la dirección IP donde está el servidor (cuando se indica *localhost* nos referimos a nuestra propia IP); luego, usuario y contraseña usados en la conexión (en el ejemplo, usuario *root* y sin contraseña; sería muy conveniente introducir otros valores en un caso real); finalmente, el nombre de la base de datos a utilizar. La conexión se guarda en el objeto `$con`. Observa como las variables en PHP siempre comienzan con el carácter `$`, además no han de declararse.

En la siguiente línea se accede a una propiedad del objeto `$con` para verificar si ha habido algún error. Observa como para acceder a las propiedades de un objeto en PHP se utilizan los caracteres `->` en lugar del carácter `.` usado en Java. Luego se configura la codificación de caracteres y se inicializa la variable `$sql` con la consulta a realizar. Solo nos interesa `puntos` y `nombre` de la tabla *puntuaciones* ordenados por `fecha`. Para concatenar dos cadenas en PHP se utiliza el carácter punto (`.`).

En el siguiente `if` verificamos si nos han pasado el parámetro `max` a través de la URL. Por ejemplo:

```
http://localhost/puntuaciones/lista.php?max=10
```

En caso de que el *array* `$_GET[ ]` contenga este parámetro, añadimos a la consulta SQL una restricción en el número de parámetros devueltos.

A continuación preparamos y ejecutamos la consulta SQL que se recogerá en la variable `$cursor`. Utilizando el método `bind_result()` asociamos los dos campos indicados en la cláusula `SELECT` con dos variables PHP. Luego recorremos todos los elementos de `$cursor` y por cada uno devolvemos una línea de texto plano

con los puntos, el nombre y un salto de línea. Más adelante intentaremos devolverlo en un formato XML. Terminamos cerrando el cursor y la conexión.

10. Abre un navegador web y escribe la siguiente dirección:

`http://localhost/puntuaciones/lista.php?max=10`

11. Es posible que el resultado se muestre en una sola línea. El navegador espera como resultado de la consulta una página HTML, y no hemos introducido en el resultado la etiqueta `<br/>` tras cada línea. Selecciona la opción *Ver código fuente de la página* para ver el resultado correctamente.

12. Crea el fichero *nueva.php* en la misma carpeta con el siguiente código:

```
<?php
   $con = new mysqli('localhost', 'root', '', 'puntuaciones');
   if ($con->connect_errno) {
      echo 'Error al conectar base de datos: ', $con->connect_error;
      exit();
   }
   $puntos = $_GET['puntos'];
   $nombre = htmlspecialchars($_GET['nombre']);
   $fecha  = $_GET['fecha'];
   $sql = $con->prepare('INSERT INTO puntuaciones VALUES (null,?, ?, ?)');
   $sql->bind_param('isi', $puntos, $nombre, $fecha);
   $sql->execute();
   echo 'OK';
   $con->close();
?>
```

13. Puedes comprobar su funcionamiento accediendo a la siguiente dirección:

`http://localhost/puntuaciones/nueva.php?puntos=3000&nombre=María&fecha=20`

14. Verifica que el nuevo elemento ha sido añadido. Puedes usar la URL:

`http://localhost/puntuaciones/lista.php?max=10`

Utilizando el servicio web PHP desde Asteroides

En este apartado vamos a realizar un cliente del servicio web diseñado en el apartado anterior para usarlo desde Asteroides.



Ejercicio: *Uso del servicio web PHP en Asteroides*

1. Abre el proyecto Asteroides.
2. Vamos a hacer el acceso a la red desde el hilo principal. Para evitar que StrictMode nos lo impida, añade el siguiente código en el método `onCreate` de *MainActivity.java*:

```
StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().
                                               permitNetwork().build());
```

3. Como en todos los ejemplos de este tema, asegúrate de que la aplicación solicita el permiso de acceso a Internet. Añade la siguiente línea en *AndroidManifest.xml*:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

4. Pasemos a implementar la interfaz `AlmacenPuntuaciones` accediendo al servidor de servicios web que acabamos de desarrollar. Para ello crea una nueva clase en la aplicación Asteroides y copia el siguiente código:

```
public class AlmacenPuntuacionesSW_PHP implements AlmacenPuntuaciones {

  public List<String> listaPuntuaciones(int cantidad) {
     List<String> result = new ArrayList<String>();
     HttpURLConnection conexion = null;
     try {
        URL url=new URL("http://158.42.146.127/puntuaciones/lista.php"
                                                  + "?max=20");
        conexion =(HttpURLConnection) url.openConnection();
        if (conexion.getResponseCode() == HttpURLConnection.HTTP_OK) {
           BufferedReader reader = new BufferedReader(new
              InputStreamReader(conexion.getInputStream()));
           String linea = reader.readLine();
           while (!linea.equals("")) {
              result.add(linea);
              linea = reader.readLine();
           }
           reader.close();
        } else {
           Log.e("Asteroides", conexion.getResponseMessage());
        }
     } catch (Exception e) {
        Log.e("Asteroides", e.getMessage(), e);
     } finally {
        if (conexion!=null) conexion.disconnect();
        return result;
     }
  }
```

El primer método se encarga de invocar la operación `lista.php` del servicio web que acabamos de implementar. Comienza definiendo la URL correspondiente al servicio web. En el código hay que reemplazar "`158.42.146.127`" por la dirección IP de tu ordenador. Recuerda que este programa lo ejecutarás desde el emulador o desde un teléfono real, y en ambos casos la IP será diferente de la de tu ordenador. Esto imposibilita utilizar como dirección `localhost`, como sí hicimos con otros clientes que ejecutábamos desde el mismo ordenador.

Una vez creada la URL se establece la conexión y mediante el método GET se manda el parámetro correspondiente.

5. Pasemos a ver el segundo método de la clase. A continuación copia el siguiente código:

```
  public void guardarPuntuacion(int puntos, String nombre, long fecha) {
```

```
        try {
            URL url=new URL("http://158.42.146.127/puntuaciones/nueva.php?"
                    + "puntos="+ puntos
                    + "&nombre="+ URLEncoder.encode(nombre, "UTF-8")
                    + "&fecha=" + fecha);
            HttpURLConnection conexion = url.openConnection();
            if (conexion.getResponseCode() == HttpURLConnection.HTTP_OK) {
                BufferedReader reader = new BufferedReader(new
                        InputStreamReader(conexion.getInputStream()));
                String linea = reader.readLine();
                if (!linea.equals("OK")) {
                    Log.e("Asteroides","Error en servicio Web nueva");
                }
            } else {
                Log.e("Asteroides", conexion.getResponseMessage());
            }
        } catch (Exception e) {
            Log.e("Asteroides", e.getMessage(), e);
        } finally {
            if (conexion!=null) conexion.disconnect();
        }
    }
}
```

La estructura de este método es similar al anterior, pero ahora llamamos a la operación `nueva.php`. Recuerda que en caso de una llamada satisfactoria, la respuesta ha de ser `OK`. Consideraremos que ha habido un error si esta no es la respuesta.

6. Puedes reemplazar "`158.42.146.127`" por la dirección IP de tu ordenador.

7. Modifica el código correspondiente para que la nueva clase pueda ser seleccionada como almacén de las puntuaciones.

8. Verifica el funcionamiento.

Creación de un servicio web en un servidor de *hosting*

Existen muchas empresas que ofrecen servicio de *hosting*, algunas incluso de forma gratuita. Por lo general, suele ser una solución más sencilla, fiable y barata que mantener nuestros propios servidores.

En este apartado aprenderemos a montar un servicio web en uno de estos servidores. Hemos elegido la empresa Hostinazo porque ofrece uno de los mejores paquetes gratuitos. Aunque en la actualidad esta empresa incrusta propaganda en su servicio gratuito, podemos encontrar otras empresas que no la incrustan. Este inconveniente podrá evitarse, dado que usaremos el *hosting* para nuestro servicio web y la propaganda incrustada no será vista por nuestros usuarios.

Los pasos del siguiente ejercicio se han preparado para el *hosting* Hostinger. No obstante, estos pasos son muy similares para otras empresas de *hosting*.



Ejercicio: *Un servicio web en un servidor de hosting*

1. Accede a la página http://www.hostinger.es. Busca la oferta "Hosting gratis".

   ***NOTA:** Este servidor ya no ofrece servicio gratuito puedes probar con https://www.hostingexperto.es/hosting-gratuito/*

2. Rellena el formulario de registro.

3. Una vez completado, te enviarán un correo para activar tu cuenta. Entra en tu correo y pulsa sobre el enlace que te indican. Si no recibes ningún correo, verifica la carpeta de *spam*.

4. Con tu usuario activado podrás crear un nuevo plan de *hosting* gratuito. Te pedirá que indiques si quieres usar tu propio dominio o usar un subdominio. Selecciona esta última opción e introduce el subdominio que prefieras. También te pedirá una nueva contraseña asociada a este *hosting*.

   Ingresa dominio y contraseña

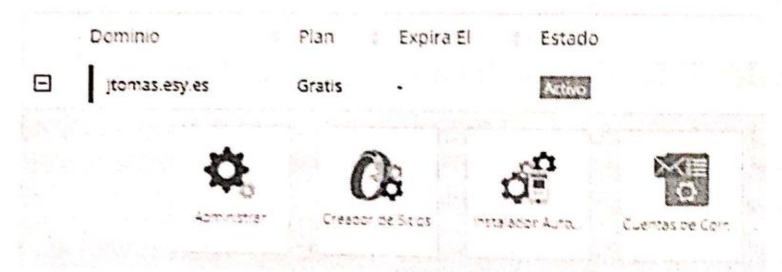
   Escoger Tipo de Dominio: Subdominio

   Subdominio jtomas .esy.es ▾

   Elije una region para el servidor * ◉ Europa (UK) ○ America del Norte (USA) ○ Asia (Hong Kong)

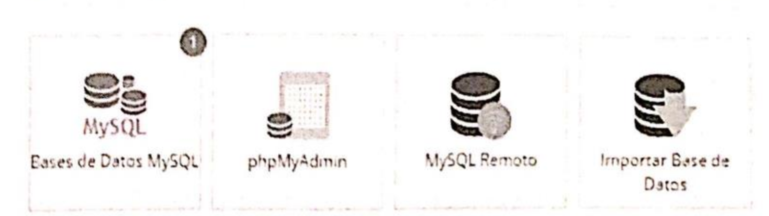
   En este ejercicio hemos usado el subdominio *jtomas.esy.es*. Tendrás que seleccionar uno diferente y acordarte de reemplazarlo en el resto del ejercicio.

5. Pasados unos minutos, ya dispondrás de tu cuenta. Refresca el navegador hasta que se muestre la siguiente información:

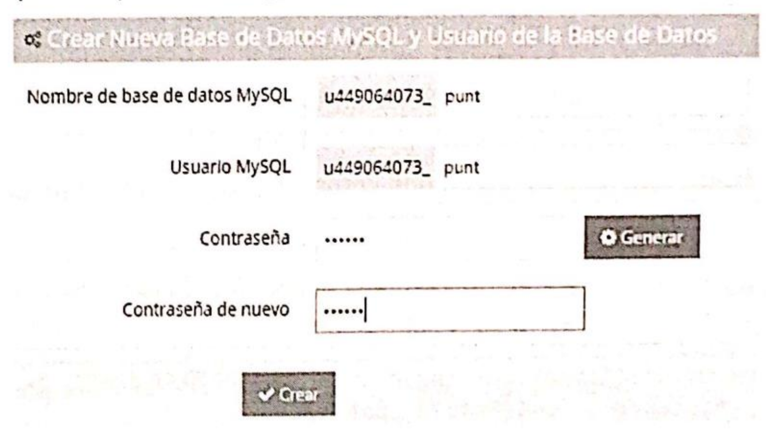


6. Introduce en un navegador tu dominio para ver que ya funciona. Por supuesto, podrás poner tu propio contenido.

7. Pulsa el botón *Administrar* para gestionar tu cuenta. Dispondrás de decenas de herramientas para gestionar diferentes aspectos de tu *hosting*. En la sección *Bases de Datos* encontrarás las siguientes opciones:



8. Selecciona la opción *Bases de Datos MySQL* y crea una nueva base de datos. Te pedirá que completes los siguientes datos:



Apunta estos datos, los vas a necesitar más tarde. Has de saber que este *hosting* gratuito solo te permite una base de datos. Pulsa *Create*.

9. Regresa a *cPanel*. Para ello pulsa el nombre de la cuenta en la barra de navegación Inicio › Hosting › jtomas.esy.es . Selecciona la herramienta *phpMyAdmin*. Te mostrará una lista con tus bases de datos. Pulsa sobre el enlace *Ingresar phpMyAdmin*.

10. Esta herramienta de gestión de bases de datos ya la utilizamos cuando instalamos el servidor en local. Para crear la tabla `puntuaciones` selecciona la lengüeta *SQL* e introduce las siguientes instrucciones en el cuadro de texto:

```
CREATE TABLE puntuaciones (
  _id INTEGER PRIMARY KEY AUTO_INCREMENT,
  puntos INTEGER, nombre TEXT, fecha BIGINT);
INSERT INTO puntuaciones (puntos, nombre, fecha) VALUES
  (10000, 'Pedro', 0),
  (20000, 'Rosa', 0);
```

11. Pulsa el botón *Continuar* para ejecutar estas sentencias SQL.

12. Recarga la página y observa como en la lista de bases de datos de la izquierda aparece la tabla *Puntuaciones*. Si pulsas sobre esta, podrás visualizar y editar su contenido.

13. Regresa a *Administrar* y en la sección *Archivos* selecciona *Administrador de Archivos 2*. Esta herramienta se situará en la raíz de tu almacenamiento.

New dir New file Upload Java Upload Install Advanced

All	Name	Type	Size
	Up..		
	logs	Directory	4096
	public_html	Directory	4096
	DO_NOT_UPLOAD_HERE	DO_NOT_UPLOAD_HERE File	0

14. Entra en la carpeta *public_html*, pulsa el botón *New dir* e introduce en el primer recuadro de texto *puntuaciones.* Entra ahora en la carpeta *puntuaciones.*
15. Pulsa el botón *Upload* y selecciona en tu ordenador el fichero *lista.php* creado en el ejercicio "Un servicio web con PHP y MySQL".
16. Sube también el fichero *nueva.php* creado en este mismo ejercicio.
17. Selecciona el archivo *lista.php* y pulsa el botón *Editar archivos.*
18. Modifica la primera línea con los datos adecuados para la base de datos creada en el punto 7. En el ejemplo mostrado sería:

```
$con = new mysqli('mysql.hostinger.es', 'u449064073_punt',
      'mi_password', 'u449064073_punt');
```

Estos cuatro parámetros corresponden al servidor MySQL que contiene la base de datos, usuario con su *password* y base de datos a la que nos conectaremos.

19. Modifica la primera línea de *nueva.php* con la misma información.
20. Ya tenemos nuestro servicio web creado. Para probarlo, introduce la siguiente dirección en un navegador, reemplazando el subdominio:

http://jtomas.esy.es/puntuaciones/lista.php

21. Utiliza la opción *Ver código fuente de la página.* Se mostrará algo similar a:

```
10000 Pedro
20000 Rosa
```

22. Utiliza la siguiente URL para verificar la inserción de datos:

http://jtomas.esy.es/puntuaciones/nueva.php?puntos=30000&nombre=Mar+Antonia&fecha=20

23. Abre el proyecto Asteroides. Edita *AlmacenPuntuacionesSW_PHP.java* y reemplaza las dos direcciones IP utilizadas por tu subdominio (`jtomas.esy.es`).
24. Ejecuta la aplicación y verifica que las puntuaciones son almacenadas en nuestro servidor de *hosting.*

Las ventajas de un servicio de *hosting* frente a un servidor propio son múltiples: el mantenimiento del servidor y su monitorización, la seguridad y las copias de seguridad dejan de ser responsabilidad nuestra. Además, el tráfico de Internet ya no ha de pasar por nuestra red. Y todo esto a precios muy reducidos o incluso gratis.

Utilizando AsyncTask de forma síncrona

Como hemos visto en este capítulo, Android impide que las operaciones con la red se realicen desde el hilo de la interfaz de usuario. Para saltarnos esta prohibición hemos desactivado `StrictMode`. Aunque lo más recomendable sería lanzar tareas asíncronas, en otros hilos, para acceder a la red.

Cuando intentamos aplicar esta segunda alternativa en Asteroides, aparece un problema: el método `AlmacenPuntuaciones.listaPuntuaciones()` ha sido diseñado para un uso síncrono; es decir, cuando se llama hay que esperar hasta que nos devuelvan el resultado, no siendo posible que el método vuelva inmediatamente, con lo que se deja la tarea pendiente. Posiblemente, habría sido interesante diseñar esta interfaz para trabajar de forma asíncrona. Por ejemplo, añadiendo el método `comienzaDescargaPuntuaciones()`, que arranca una tarea para la descarga y devuelve inmediatamente el control sin devolver nada. Desde esta tarea se podría lanzar un evento cuando se dispusiera de las puntuaciones.

En este ejercicio no vamos a cambiar el diseño de esta interfaz. En lugar de ello, vamos a introducir un `AsyncTask` dentro de `listaPuntuaciones()` y no retornaremos de este método hasta que termine y ya tengamos las puntuaciones. El siguiente esquema muestra este planteamiento:

Hilo principal	**Hilo secundario**
`listaPuntuaciones() {`	
`tarea.execute();` →	`Resultado doInBackground () {`
↓	`...`
`return tarea.get();` ←	`return lista;`
`}`	`}`

Trabajando de esta manera realizamos el acceso a la red desde un hilo secundario. Por lo tanto, `StrictMode` no se quejará. Pero es muy importante que entiendas que realmente no hemos resuelto nada. Esta forma de trabajar bloquea igualmente el hilo de la interfaz de usuario. Es decir, un método `execute()` seguido de un `get()` es equivalente a llamar la tarea de una forma síncrona.

Entonces, ¿qué ventaja tiene usar un `AsyncTask`? En el método `get()` vamos a poder fijar un tiempo máximo a la tarea, por ejemplo `get(4, TimeUnit.SECONDS)`. Pasado este tiempo, informamos al usuario de que el servidor no responde y continuamos.



Ejercicio: *Uso síncrono de AsyncTask para acceso al servicio web PHP de puntuaciones*

1. En el proyecto Asteroides, copia la clase `AlmacenPuntuacionesSW_PHP` en una nueva clase y llámala `AlmacenPuntuacionesSW_PHP_AsyncTask`.
2. Introduce las siguientes líneas al comienzo de la nueva clase, reemplazando el método `listaPuntuaciones()`:

```
private Context contexto;

public AlmacenPuntuacionesSW_PHP_AsyncTask(Context contexto) {
   this.contexto = contexto;
}

public List<String> listaPuntuaciones(int cantidad) {
   try {
      TareaLista tarea = new TareaLista();
      tarea.execute(cantidad);
      return tarea.get(4, TimeUnit.SECONDS);
   } catch (TimeoutException e) {
      Toast.makeText(contexto, "Tiempo excedido al conectar",
            Toast.LENGTH_LONG).show();
   } catch (CancellationException e) {
      Toast.makeText(contexto, "Error al conectar con servidor",
            Toast.LENGTH_LONG).show();
   } catch (Exception e) {
      Toast.makeText(contexto, "Error con tarea asíncrona",
            Toast.LENGTH_LONG).show();
   }
   return new ArrayList<String>();
}

private class TareaLista extends AsyncTask<Integer, Void, List<String>>{
   @Override
   protected List<String> doInBackground(Integer... cantidad){
      //Copia el código que antes estaba en listaPuntuaciones()
   }
}
```

Empezamos añadiendo un constructor a la clase para indicar el contexto donde se ejecuta. Esto nos permitirá introducir `Toast()` en la clase.

Para obtener la lista de puntuaciones desde un nuevo hilo se ha creado un descendiente de `AsyncTask` que toma como parámetro de entrada un entero con la cantidad máxima de puntuaciones a obtener y nos devuelve una lista de `String`. El código de la tarea a realizar es casi idéntico al usado en el ejercicio anterior. Por esta razón no se ha incluido. En el siguiente punto se indica lo único que tendrás que cambiar. Para usar esta nueva clase se ha instanciado el objeto `tarea`.

En `listaPuntuaciones()` comenzamos instanciando un objeto de la clase `TareaLista`. El método `execute()` es utilizado para pasar los parámetros de entrada y arrancar la tarea. El método `get()` espera a que la tarea concluya y nos devuelve su salida. Como se ha comentado en el capítulo 5, hay que usar este método con cuidado, dado que bloquea el hilo de la interfaz de usuario. Sin embargo, dado que mientras estamos esperando la respuesta del servidor el usuario no puede realizar ninguna interacción, no va a suponer ningún problema. Para asegurarnos de que no nos quedamos bloqueados un tiempo excesivo, usamos una de las sobrecargas del método `get()`, que nos permite indicar el tiempo máximo a esperar y la unidad en que medimos este tiempo. En caso de sobrepasar este tiempo, se generará una excepción `TimeoutException`, que se

```
private Context contexto;

public AlmacenPuntuacionesSW_PHP_AsyncTask(Context contexto) {
   this.contexto = contexto;
}

public List<String> listaPuntuaciones(int cantidad) {
   try {
      TareaLista tarea = new TareaLista();
      tarea.execute(cantidad);
      return tarea.get(4, TimeUnit.SECONDS);
   } catch (TimeoutException e) {
      Toast.makeText(contexto, "Tiempo excedido al conectar",
            Toast.LENGTH_LONG).show();
   } catch (CancellationException e) {
      Toast.makeText(contexto, "Error al conectar con servidor",
            Toast.LENGTH_LONG).show();
   } catch (Exception e) {
      Toast.makeText(contexto, "Error con tarea asíncrona",
            Toast.LENGTH_LONG).show();
   }
   return new ArrayList<String>();
}

private class TareaLista extends AsyncTask<Integer, Void, List<String>>{
   @Override
   protected List<String> doInBackground(Integer... cantidad){
      //Copia el código que antes estaba en listaPuntuaciones()
   }
}
```

Empezamos añadiendo un constructor a la clase para indicar el contexto donde se ejecuta. Esto nos permitirá introducir `Toast()` en la clase.

Para obtener la lista de puntuaciones desde un nuevo hilo se ha creado un descendiente de `AsyncTask` que toma como parámetro de entrada un entero con la cantidad máxima de puntuaciones a obtener y nos devuelve una lista de `String`. El código de la tarea a realizar es casi idéntico al usado en el ejercicio anterior. Por esta razón no se ha incluido. En el siguiente punto se indica lo único que tendrás que cambiar. Para usar esta nueva clase se ha instanciado el objeto `tarea`.

En `listaPuntuaciones()` comenzamos instanciando un objeto de la clase `TareaLista`. El método `execute()` es utilizado para pasar los parámetros de entrada y arrancar la tarea. El método `get()` espera a que la tarea concluya y nos devuelve su salida. Como se ha comentado en el capítulo 5, hay que usar este método con cuidado, dado que bloquea el hilo de la interfaz de usuario. Sin embargo, dado que mientras estamos esperando la respuesta del servidor el usuario no puede realizar ninguna interacción, no va a suponer ningún problema. Para asegurarnos de que no nos quedamos bloqueados un tiempo excesivo, usamos una de las sobrecargas del método `get()`, que nos permite indicar el tiempo máximo a esperar y la unidad en que medimos este tiempo. En caso de sobrepasar este tiempo, se generará una excepción `TimeoutException`, que se

procesa en la sección *catch*. También se recoge la posibilidad de que ocurra una excepción de cancelación de tarea. Al final del ejercicio se añade el método adecuado para cancelar si ocurre algún tipo de error.

3. Reemplaza el código `+ "?max="+cantidad);` por `+ "?max="+cantidad[0]);`. Aunque esta tarea solo necesita un entero, la mecánica de AsyncTask hace que se nos pase un *array* de enteros.

4. Introduce las siguientes líneas, reemplazando el método `guardarPuntuacion()`:

```
public void guardarPuntuacion(int puntos,String nombre, long fecha){
   try {
      TareaGuardar tarea = new TareaGuardar();
      tarea.execute(String.valueOf(puntos), nombre,
            String.valueOf(fecha));
      tarea.get(4, TimeUnit.SECONDS);
   } catch (TimeoutException e) {
      Toast.makeText(contexto, "Tiempo excedido al conectar",
            Toast.LENGTH_LONG).show();
   } catch (CancellationException e) {
      Toast.makeText(contexto, "Error al conectar con servidor",
            Toast.LENGTH_LONG).show();
   } catch (Exception e) {
      Toast.makeText(contexto, "Error con tarea asíncrona",
            Toast.LENGTH_LONG).show();
   }
}

private class TareaGuardar extends AsyncTask<String, Void, Void> {
   @Override
   protected Void doInBackground(String... param) {
      try {
         URL url = new URL(
                  "http://jtomas.hostinazo.com/puntuaciones/nueva.php"
                  + "?puntos=" + param[0] + "&nombre="
                  + URLEncoder.encode(param[1], "UTF-8")
                  + "&fecha=" + param[2]);
         //Copia el código que antes estaba en guardarPuntuaciones
         return null;
   }
}
```

La mecánica para llamar al servicio web que almacena una nueva puntuación es similar al anterior. La única diferencia está en las clases que parametriza el `AsyncTask`. Ahora, como entrada, hay que introducir tres `strings`: puntos, nombre y fecha de la puntuación. Además, la tarea no nos devuelve ninguna información.

5. Busca en la clase las apariciones de `Log.e(...);` y añade en la fila inferior `cancel(true);`. En total tienes que añadir 5 líneas. Con esto indicamos que queremos que se cancele la tarea en caso de error.

6. Modifica la clase *MainActivity.java* y las *res/values/arrays.xml* para que el nuevo tipo de almacenamiento pueda ser seleccionado.

7. Modifica el código correspondiente para que la nueva clase pueda ser seleccionada como almacén de las puntuaciones.
8. Verifica el funcionamiento.
9. Para verificar que su comportamiento es robusto ante errores en la red, desconecta el acceso a Internet del dispositivo y verifica que al listar las puntuaciones te indica: "Error al conectar con servidor". Prueba a introducir la llamada `sleep(5)` en el fichero *lista.php* del servidor. Con esto se añade un retardo de 5 segundos en la respuesta. Verifica que al listar las puntuaciones te indica: "Tiempo excedido al conectar".

10.4.4. Comparativa *sockets*/servicios web

En este capítulo hemos utilizado dos alternativas, *sockets* y servicios web, para resolver un mismo problema. En la mayoría de los casos es más recomendable utilizar servicios web. Veamos las ventajas de un servicio web frente a un servidor de *sockets*:

La principal ventaja de los servicios web es la **claridad de diseño**. Para acceder al servicio resulta mucho más sencillo utilizar un método estándar muy conocido basado en URL, en lugar de tener que crear nuestro propio protocolo.

Otra ventaja es el **aprovechamiento de las cabeceras HTTP**. Como se comentó en el apartado anterior, el protocolo HTTP incorpora una serie de cabeceras para ofrecer información adicional en el intercambio. Mediante estas cabeceras podemos controlar aspectos muy importantes, como solicitar la autentificación del cliente, utilizar un modo seguro de transferencia (*https*), definir el tipo de información transmitida o controlar si queremos que las peticiones a nuestros servicios sean recordadas en la caché del cliente y por cuánto tiempo.

El uso de **servidores comerciales** en los servicios web nos proporciona grandes ventajas, que sería complejo implementar en nuestro servidor de *sockets*. Por ejemplo, en un servidor web como Apache se incluye la seguridad, la escalabilidad y facilidades de gestión.

Ambos servicios han de ofrecerse a través de un puerto. Los servicios web suelen utilizar el **mismo puerto que los servidores web**, el 80. Esto presenta la ventaja de tratarse de un puerto que raramente es filtrado por los cortafuegos. Esta ventaja también puede utilizarse en un servidor por *sockets* si le asignamos este puerto. Pero en este caso, ya no podrás instalar en la misma máquina un servidor web.



Preguntas de repaso: *Servicios web*

ANEXO A.

Diálogos de fecha y hora

Los cuadros de diálogo se han introducido en el ejercicio "**Un cuadro de diálogo para indicar el *id* de lugar**", donde hemos aprendido a realizar un cuadro de diálogo personalizado. En este apartado aprenderemos a utilizar cuadros de diálogo específicos para trabajar con fechas y horas. Empezaremos introduciendo algunos conceptos y clases que nos ayudarán a trabajar con este tipo de información.

Clases para trabajar con fechas en Java

Clase Date[49]

La clase `Date` Representa un instante en el tiempo con una precisión de milisegundos. Se utiliza un sistema de medición del tiempo independiente de la zona horaria, conocido como UTC (tiempo universal coordinado). El estándar de medición de tiempo UTC utiliza el tiempo en el meridiano de Greenwich independientemente de donde nos encontremos. De esta forma, se evitan los problemas que aparecen cuando se comunican dos sistemas con mediciones locales de tiempo diferentes.

Para representar un instante de tiempo se suele utilizar la codificación conocida como tiempo Unix. Esta codificación consiste en medir el número de milisegundos trascurridos desde el 1 de enero de 1970. Para almacenar este valor se utiliza un entero de 64 bits. En Java, la palabra reservada `long` representa un entero de este tipo. En Android, si quieres obtener el tiempo actual en este formato utiliza el método `currentTimeMillis()` de la clase `System`.

```
long ahora = System.currentTimeMillis();
Date fecha = new Date(ahora);
```

[49] http://developer.android.com/reference/java/util/Date.html

Clase DateFormat[50]

La clase Date está pensada para contar el tiempo de forma universal en toda la Tierra, de forma que sea sencilla de manipular por una máquina. Sin embargo, las personas utilizamos una medición del tiempo que depende de la zona horaria donde estemos o incluso de si el país donde estemos utiliza el horario de verano. Cuando tengas que mostrar o solicitar una fecha a una persona, deberás utilizar la representación del tiempo a la que está acostumbrada. En este caso, la clase abstracta DateFormat o su descendiente SimpleDateFormat[51] te serán de gran ayuda para este propósito.

A continuación se muestra un ejemplo sencillo:

```
DateFormat df = new SimpleDateFormat("dd/MM/yy");
String salida = df.format(fecha);
```

Clase Calendar[52]

Como hemos comentado, la clase Date utiliza internamente un simple entero para representar un instante de tiempo. Por el contrario, los humanos nos complicamos algo más, dado que usamos la combinación de varios campos, como año, mes, día, hora, minuto, etc. Utiliza la clase Calendar para obtener estos campos desde un objeto Date. A diferencia de la clase Date, la clase Calendar depende de la configuración local del dispositivo (*locale*). Para obtener la fecha actual según la representación local del dispositivo, utiliza el método getInstance():

```
Calendar calendario = Calendar.getInstance();
calendario.setTimeInMillis(ahora);
int hora = calendario.get(Calendar.HOUR_OF_DAY);
int minuto = calendario.get(Calendar.MINUTE);
```

La clase Calendar es una clase abstracta, que en principio te permitiría trabajar con cualquier clase de calendario (como el calendario maya o el musulmán). No obstante, el calendario usado oficialmente en casi todo el mundo es el calendario gregoriano, definido en la clase GregorianCalendar.



Ejercicio: *Añadiendo un diálogo de selección de hora*

Un cuadro de diálogo es un tipo de ventana emergente que solicita al usuario de la aplicación algún tipo de información, antes de realizar algún proceso. Este tipo de ventanas no suele ocupar la totalidad de la pantalla. En la aplicación Mis Lugares hemos utilizado diálogos en dos ocasiones: para indicar el *id* a mostrar y para confirmar el borrado de un lugar. En este ejercicio aprenderemos a hacer un diálogo más complejo, que permite modificar la hora y los minutos.

---

[50] http://developer.android.com/reference/java/text/DateFormat.html

[51] http://developer.android.com/reference/java/text/SimpleDateFormat.html

[52] http://developer.android.com/reference/java/util/Calendar.html



1. Abre el layout *vista_lugar.xml* y localiza el `<imageView>` que indica la hora asociada al lugar. Añade el atributo marcado:

```
<ImageView
    android:id="@+id/icono_hora"
    android:layout_width="40dp"
    android:layout_height="40dp"
    android:contentDescription="logo de la hora"
    android:src="@android:drawable/ic_menu_recent_history" />
```

2. Abre la clase `VistaLugarActivity` y añade en el método `onCreate()` el siguiente código:

```
findViewById(R.id.icono_hora).setOnClickListener(
   new OnClickListener() {
      public void onClick(View view) { usoLugarFecha.cambiarHora(pos); }
});
findViewById(R.id.hora).setOnClickListener(
   new OnClickListener() {
      public void onClick(View view) { usoLugarFecha.cambiarHora(pos); }
});
```

```
icono_hora.setOnClickListener { usoLugarFecha.cambiarHora(pos) }
hora.setOnClickListener { usoLugarFecha.cambiarHora(pos) }
```

NOTA: *Selecciona el paquete android.view.OnClickListener.*

3. Como acabas de ver, vamos a crear un nuevo caso de uso para cambiar la hora de un lugar. La clase `CasosUsoLugar` empieza a ser demasiado grande. Podría ser interesante dividirla en tres partes: Operaciones de tipo CRUD (altas, bajas, modificaciones…), fotografías y de fecha y hora. De momento vamos a añadir las operaciones de fecha y hora en la clase `CasosUsoLugarFecha`:

```
public class CasosUsoLugarFecha {
   protected AppCompatActivity actividad;
   protected LugaresBDAdapter lugares;

   public CasosUsoLugarFecha(AppCompatActivity actividad,
                             LugaresBDAdapter lugares){
```

```
        this.actividad = actividad;
        this.lugares = lugares;
    }
}
```

```
class CasosUsoLugarFecha(val actividad: AppCompatActivity,
                         val lugares: LugaresBDAdapter) {
}
```

4. Añade en la nueva clase:

```
private int pos =-1;
private Lugar lugar;

public void cambiarHora(int pos) {
   lugar = lugares.elementoPos(pos);
   this.pos = pos;
   DialogoSelectorHora dialogo = new DialogoSelectorHora();
   dialogo.setOnTimeSetListener(this);
   Bundle args = new Bundle();
   args.putLong("fecha", lugar.getFecha());
   dialogo.setArguments(args);
   dialogo.show(actividad.getSupportFragmentManager(), "selectorHora");
}
```

```
var pos: Int = -1
lateinit var lugar: Lugar

fun cambiarHora(pos: Int, textView: TextView) {
   lugar = lugares.elementoPos(pos)
   this.pos = pos
   val dialogo = DialogoSelectorHora()
   dialogo.setOnTimeSetListener(this)
   val args = Bundle();
   args.putLong("fecha", lugar.fecha)
   dialogo.setArguments(args)
   dialogo.show(actividad.supportFragmentManager, "selectorHora")
}
```

Este método se ejecutará cuando se pulse sobre la hora. Su objetivo es mostrar un cuadro de diálogo para que el usuario pueda modificar la hora asociada al lugar. Los parámetros son: el `pos` del lugar a modificar y el `TextView` donde escribiremos la nueva hora. Comenzamos escribiendo dos variables donde recordaremos la información que estamos modificando, tras volver del diálogo.

Continuamos creando un nuevo diálogo y luego le asignamos el escuchador a nuestra propia clase. De esta forma, cuando el usuario cambie la hora se llamará a un método de nuestra clase. Este método lo crearemos en uno de los puntos siguientes. A este diálogo le pasamos como argumento la fecha del lugar en un `long`. Finalmente, mostramos el diálogo llamando al método `show()`. Este método utiliza dos parámetros: el manejador de *fragments* y una etiqueta que identificará el cuadro de diálogo.

5. Crea la siguiente clase en el paquete `presentacion`:

```
public class DialogoSelectorHora extends DialogFragment {

    private OnTimeSetListener escuchador;

    public void setOnTimeSetListener(OnTimeSetListener escuchador) {
        this.escuchador = escuchador;
    }

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
        Calendar calendario = Calendar.getInstance();
        Bundle args = this.getArguments();
        if (args != null) {
            long fecha = args.getLong("fecha");
            calendario.setTimeInMillis(fecha);
        }
        int hora = calendario.get(Calendar.HOUR_OF_DAY);
        int minuto = calendario.get(Calendar.MINUTE);
        return new TimePickerDialog(getActivity(), escuchador, hora,
                minuto, DateFormat.is24HourFormat(getActivity()));
    }
}
```

```
class DialogoSelectorHora : DialogFragment() {

   private var escuchador: OnTimeSetListener? = null

   fun setOnTimeSetListener(escuchador: OnTimeSetListener) {
      this.escuchador = escuchador
   }

   override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
      val calendario = Calendar.getInstance()
      val fecha = arguments?.getLong("fecha")?:System.currentTimeMillis()
      calendario.setTimeInMillis(fecha)
      val hora = calendario.get(Calendar.HOUR_OF_DAY)
      val minuto = calendario.get(Calendar.MINUTE)
      return TimePickerDialog(
         getActivity(), escuchador, hora,
         minuto, DateFormat.is24HourFormat(getActivity())
      )
   }
}
```

NOTA: *Pulsa* ***Alt-Intro*** *para añadir los imports automáticamente. Algunas clases se encuentran en varios paquetes, por lo que te preguntará. Utiliza:*

androidx.fragment.app.DialogFragment

android.text.format.DateFormat

Esta clase extiende `DialogFragment`, que define un *fragment* que muestra una ventana de diálogo flotante sobre la actividad. El control del cuadro de diálogo debe hacerse siempre a través de los métodos de la API, nunca directamente.

Para definir un nuevo `DialogFragment` se puede sobrescribir `onCreateView()` para indicar el contenido del diálogo. Alternativamente, se puede sobrescribir `onCreateDialog()` para crear un diálogo totalmente personalizado, como hacemos en este ejercicio. En este método hay que devolver un objeto `Dialog`, que es el que se mostrará.

Creamos un objeto `Calendar` y si nos han pasado una fecha se la asignamos. En caso contrario, la fecha corresponderá con la actual. Luego extraemos la hora y los minutos del calendario.

Finalmente creamos un nuevo diálogo de la clase `TimePickerDialog`. Se trata de un tipo de diálogo definido en el sistema que nos permite seleccionar horas y minutos. En su constructor indicamos cuatro parámetros: el contexto, un escuchador al que llamará cuando se seleccione la hora, la hora y los minutos que se mostrarán al inicio y un valor booleano que indica si trabajamos con formato de 24 horas o de 12. En el código se usa el valor definido en nuestro contexto.

6. Haz que `CasosUsoLugarFecha` implemente la siguiente interfaz:

```
public class CasosUsoLugarFecha
                              implements TimePickerDialog.OnTimeSetListener {
```

```
class CasosUsoLugarFecha(…) : TimePickerDialog.OnTimeSetListener {
```

7. Añade la siguiente función:

```
@Override public void onTimeSet(TimePicker vista, int hora, int minuto) {
   Calendar calendario = Calendar.getInstance();
   calendario.setTimeInMillis(lugar.getFecha());
   calendario.set(Calendar.HOUR_OF_DAY, hora);
   calendario.set(Calendar.MINUTE, minuto);
   lugar.setFecha(calendario.getTimeInMillis());
   lugares.actualizaPosLugar(pos, lugar);
   TextView textView = actividad.findViewById(R.id.hora);
   textView.setText(DateFormat.getTimeInstance().format(
                                          new Date(lugar.getFecha())));
}
```

```
override fun onTimeSet(vista: TimePicker?, hora: Int, minuto: Int) {
   val calendario = Calendar.getInstance()
   calendario.setTimeInMillis(lugar.fecha)
   calendario.set(Calendar.HOUR_OF_DAY, hora)
   calendario.set(Calendar.MINUTE, minuto)
   lugar.fecha = calendario.getTimeInMillis()
   lugares.actualizaPosLugar(pos, lugar)
   val textView = actividad.findViewById<TextView>(R.id.hora)
   textView.text= DateFormat.getTimeInstance().format(Date(lugar.fecha))
}
```

En el punto anterior hemos indicado que nuestra clase actuará como escuchador cuando se seleccione una hora en el cuadro de diálogo. Como consecuencia, se llamará a este método. Se nos pasan tres parámetros. En este caso nos interesan la hora y los minutos seleccionados. Para cambiar esta información en la fecha

asociada al lugar, comenzamos creando un objeto `Calendar` y lo inicializamos con la fecha que tiene el lugar. Luego le modificamos la hora y los minutos según los parámetros que nos han indicado. Hay que aclarar que el resto de la fecha, como el día o el mes, no se modificarán. La nueva fecha es introducida en el objeto `lugar` y a continuación actualizamos la base de datos.

Para modificar el `TextView` de la hora, comenzamos creando un formato de fecha, donde se visualizará la hora y los minutos separado por dos puntos. Para convertir la fecha correctamente hay que conocer la zona horaria definida en el sistema. Esto se consigue con `java.util.Locale.getDefault()`. Finalmente usamos este formato sobre un objeto `Date` para cambiar el contenido del `TextView`.

8. En `VistaLugarActivity` crea la variable `usoLugarFecha` de la clase `CasosUsoLugarFecha` de igual forma como se ha creado `usoLugar`.

9. Ejecuta la aplicación y verifica el resultado.



Práctica: *Añadiendo un diálogo de selección de fecha*

Podrías crear un cuadro de diálogo para modificar la fecha asociada al lugar (día, mes y año). Has de realizar los mismos pasos que en el ejercicio anterior, pero ahora se basará en el diálogo siguiente:



En este caso tendrás que usar un diálogo de la clase `DatePickerDialog`:

```
DatePickerDialog(actividad , escuchador, año, mes, dia);
```

El escuchador ha de implementar la interfaz `OnDateSetListener` y esta interfaz define el siguiente método:

```
@Override
public void onDateSet(DatePicker view, int año, int mes, int dia) {…}
```

Finalmente, utiliza el siguiente formato para representarlo:

```
DateFormat formato = DateFormat.getDateInstance();
```

En este caso no se define un formato concreto como en el ejercicio anterior, si no que se selecciona el definido en el sistema para representar una fecha. De esta forma, el formato será el que ha configurado el usuario en el dispositivo.



Solución:

Añade en `VistaLugarActivity`, dentro de `onCreate()`:

```
findViewById(R.id.icono_fecha).setOnClickListener(
   new OnClickListener() {
      public void onClick(View view) { usoLugarFecha.cambiarFecha(pos); }
});
findViewById(R.id.fecha).setOnClickListener(
   new OnClickListener() {
      public void onClick(View view) { usoLugarFecha.cambiarFecha(pos); }
});
```

```
icono_fecha.setOnClickListener { usoLugarFecha.cambiarFecha(pos) }
fecha.setOnClickListener { usoLugarFecha.cambiarFecha(pos) }
```

En la clase `CasosUsoLugarFecha` añade la interfaz y las dos funciones indicadas.

```
public class CasosUsoLugarFecha implements
  TimePickerDialog.OnTimeSetListener, DatePickerDialog.OnDateSetListener {

public void cambiarFecha(int pos) {
   lugar = lugares.elementoPos(pos);
   this.pos = pos;
   DialogoSelectorFecha dialogo = new DialogoSelectorFecha();
   dialogo.setOnDateSetListener(this);
   Bundle args = new Bundle();
   args.putLong("fecha", lugar.getFecha());
   dialogo.setArguments(args);
   dialogo.show(actividad.getSupportFragmentManager(),"selectorFecha");
}

@Override
public void onDateSet(DatePicker view, int año, int mes, int dia) {
   Calendar calendario = Calendar.getInstance();
   calendario.setTimeInMillis(lugar.getFecha());
   calendario.set(Calendar.YEAR, año);
   calendario.set(Calendar.MONTH, mes);
   calendario.set(Calendar.DAY_OF_MONTH, dia);
   lugar.setFecha(calendario.getTimeInMillis());
   lugares.actualizaPosLugar(pos, lugar);
   TextView textView = actividad.findViewById(R.id.fecha);
```

```
   textView.setText(DateFormat.getDateInstance().format(
                                              new Date(lugar.getFecha())));
}
```

```
class CasosUsoLugarFecha(…) : TimePickerDialog.OnTimeSetListener,
                              DatePickerDialog.OnDateSetListener{

fun cambiarFecha(pos: Int) {
   lugar = lugares.elementoPos(pos)
   this.pos = pos
   val dialogo = DialogoSelectorFecha()
   dialogo.setOnDateSetListener(this)
   val args = Bundle()
   args.putLong("fecha", lugar.fecha)
   dialogo.setArguments(args)
   dialogo.show(actividad.supportFragmentManager, "selectorFecha")
}

override fun onDateSet(view: DatePicker, año: Int, mes: Int, dia: Int) {
   val calendario = Calendar.getInstance()
   calendario.timeInMillis = lugar.fecha
   calendario.set(Calendar.YEAR, año)
   calendario.set(Calendar.MONTH, mes)
   calendario.set(Calendar.DAY_OF_MONTH, dia)
   lugar.fecha = calendario.timeInMillis
   lugares.actualizaPosLugar(pos, lugar)
   val textView = actividad.findViewById<TextView>(R.id.fecha)
   textView.text =
          java.text.DateFormat.getDateInstance().format(Date(lugar.fecha))
}
```

Crea la clase `DialogoSelectorFecha`:

```
public class DialogoSelectorFecha extends DialogFragment {

   private OnDateSetListener escuchador;

   public void setOnDateSetListener(OnDateSetListener escuchador) {
      this.escuchador = escuchador;
   }
   @Override public Dialog onCreateDialog(Bundle savedInstanceState) {
      Calendar calendario = Calendar.getInstance();
      Bundle args = this.getArguments();
      if (args != null) {
         long fecha = args.getLong("fecha");
         calendario.setTimeInMillis(fecha);
      }
      int año = calendario.get(Calendar.YEAR);
      int mes = calendario.get(Calendar.MONTH);
      int dia = calendario.get(Calendar.DAY_OF_MONTH);
      return new DatePickerDialog(getActivity(),escuchador,año,mes,dia);
   }
}
```

```
class DialogoSelectorFecha : DialogFragment() {

   private var escuchador: OnDateSetListener? = null

   fun setOnDateSetListener(escuchador: OnDateSetListener) {
      this.escuchador = escuchador
   }

   override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
      val calendario = Calendar.getInstance()
      val fecha = getArguments()?.getLong("fecha")?:0L
      calendario.setTimeInMillis(fecha)
      val año = calendario.get(Calendar.YEAR)
      val mes = calendario.get(Calendar.MONTH)
      val dia = calendario.get(Calendar.DAY_OF_MONTH)
      return DatePickerDialog(getActivity()!!.applicationContext,
                              escuchador, año, mes, dia)
   }
}
```

ANEXO B.

Fragments

Con la popularización de las tabletas surgió el problema de desarrollar simultáneamente una aplicación para ser ejecutada tanto en un móvil como en una tableta. En otros sistemas, como iOS, se decidió que el desarrollador tenía que implementar dos aplicaciones diferentes. Android siguió con la estrategia de usar recursos alternativos para adaptarse a los diferentes tamaños de pantalla. Las herramientas vistas hasta ahora no resultan suficientes: cuando se diseña una interfaz de usuario específica para una tableta, no solo es preciso adaptar el tamaño de letra o los márgenes, si no que también es necesario reestructurar cómo se muestra la información en pantalla. En una tableta pueden caber muchos elementos de diseño al mismo tiempo, mientras que en un móvil estamos más limitados. Por ejemplo, podríamos diseñar dos elementos de la interfaz de usuario: uno que nos permitiera elegir entre una lista de lugares y otro que mostrara los detalles de uno de esos lugares. En una tableta se podrían mostrar ambos elementos a la vez, mientras que en un móvil tendríamos que mostrar primero uno y luego el otro.



Vídeo[tutorial]: *Los* fragments *en Android*

Para resolver este problema, en la versión 3.0 de Android se introdujeron los *fragments*. Los *fragments* son bloques de interfaz de usuario que pueden utilizarse en diferentes sitios, simplificando así la composición de una interfaz de usuario. Los *fragments* nos permiten diseñar y crear cada uno de los elementos de nuestra aplicación por separado. Luego, dependiendo del tamaño de pantalla disponible, mostraremos uno solo o más de uno a la vez.



Figura 6: *Uso de* fragments *en tableta y móvil.*

Es importante resaltar que no cambia el papel de las actividades. Sigue siendo el elemento básico que representa cada pantalla de una aplicación y nos permite navegar por ella. La novedad introducida es que cuando diseñemos una actividad, esta puede estar formada por uno o más *fragments*.

Cuando diseñemos un *fragment*, este ha de gestionarse a sí mismo, recibiendo eventos de entrada y modificando su vista sin necesidad de que la actividad que lo contiene intervenga. De esta forma, el *fragment* se podrá utilizar en diferentes actividades sin tener que modificar el código.

Los *fragments* son muy importantes, dado que a partir de ahora Android los utiliza como elemento base en el diseño de la interfaz de usuario. Por ejemplo, la última versión de Google Maps y la visualización de preferencias de usuario se basan en *fragments*. El problema es que esta característica aparece en una versión que todavía no está disponible en muchos dispositivos. Para resolver este problema se ha creado una librería de compatibilidad para poder utilizar *fragments* en versiones anteriores a la 3.0. Esta librería se incluye de manera automática en un proyecto, siempre que el requerimiento mínimo de SDK sea inferior al nivel 11 (3.0), pero lo desarrollemos con una versión igual o superior a la 3.0 (*Target SDK*). Para verificar que así sea, abre el proyecto creado en el ejercicio anterior. Observa como esta librería se incluye en *libs/android-support-v4.jar.*

Cada *fragment* ha de implementarse en una clase diferente. Esta clase tiene una estructura similar a la de una actividad, pero con algunas diferencias. La primera es que esta clase tiene que extender `Fragment`. El ciclo de vida es muy parecido al de una actividad; sin embargo, dispone de unos cuantos eventos más, que le indican cambios en su estado con respecto a la actividad que lo contiene. El ciclo de vida de un *fragment* va asociado al de la actividad que lo contiene (por ejemplo, si la actividad es destruida, todos los *fragments* que contiene son destruidos); pero también es posible destruir un *fragment* sin modificar el estado de la actividad.

Los *fragments* suelen mostrar una vista (aunque esto no es imprescindible). Es recomendable definir esta vista en un fichero XML de recursos. Por lo tanto, para

crear un *fragment* usaremos una clase Java para definir su comportamiento y un fichero XML para definir su apariencia.

Los *fragments* se pueden introducir en una actividad de dos formas diferentes: por código o desde XML. Ambas formas tienen sus ventajas y sus inconvenientes. Introducir un *fragment* desde XML es más sencillo. Además, el diseño queda diferenciado del código, simplificando el trabajo del diseñador. Sin embargo, trabajar de esta forma tiene un inconveniente: una vez introducido ya no podremos reemplazar el *fragment* por otro. Por lo tanto, un *fragment* añadido desde XML será siempre estático. Si lo añadimos desde código, ganamos la posibilidad de intercambiar el *fragment* por otro. En los siguientes ejercicios veremos cómo añadir *fragments* desde XML.



Ejercicio: *Un primer* fragment

En este ejercicio modificaremos la aplicación Mis Lugares, pero ahora trabajando con un *fragment*. Su funcionalidad será idéntica. La ventaja de definir *fragments* en vez de actividades es que podemos mostrar varios *fragments* a la vez en la pantalla, pero no varias actividades.

1. En este apartado vamos a realizar un número importante de modificaciones y es posible que algo salga mal. Puede ser un buen momento para realizar una copia del proyecto actual. Así, siempre dispondremos de una versión operativa. Para ello, desde el explorador de ficheros de tu sistema operativo, realiza una copia de la carpeta que contiene el proyecto. Para acceder rápidamente a esta carpeta desde el explorador del proyecto, pulsa en *app* con el botón derecho y selecciona *Show in Explorer*.
2. En este ejercicio vamos a mostrar en un *fragment* lo que antes se mostraba en `MainActivity`. Por lo tanto, podemos reutilizar su *layout* en XML para nuestro *fragment*. Copia el fichero *content_main.xml* en *fragment_selector.xml*. Desde el explorador del proyecto usa *Ctrl-C* y *Ctrl-V*.
3. Ahora nos falta definir la clase para el *fragment*. Crea una nueva clase llamada `SelectorFragment` y rellénala con el siguiente código:

```
public class SelectorFragment extends Fragment {
   private LugaresBDAdapter lugares;
   private AdaptadorLugaresBD adaptador;
   private CasosUsoLugar usoLugar;
   private RecyclerView recyclerView;

   @Override
   public View onCreateView(LayoutInflater inflador, ViewGroup contenedor,
                            Bundle savedInstanceState) {
      View vista = inflador.inflate(R.layout.fragment_selector,
            contenedor, false);
      recyclerView = vista.findViewById(R.id.recyclerView);
      return vista;
   }
```

```
    @Override
    public void onActivityCreated(Bundle state) {
        super.onActivityCreated(state);
        lugares = ((Aplicacion) getActivity().getApplication()).lugares;
        adaptador = ((Aplicacion) getActivity().getApplication()).adaptador;
        usoLugar = new CasosUsoLugar(getActivity(), lugares);
        recyclerView.setHasFixedSize(true);
        recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
        recyclerView.setAdapter(adaptador);
        adaptador.setOnItemClickListener(new View.OnClickListener() {
            @Override public void onClick(View v) {
                int pos = (Integer)(v.getTag());
                usoLugar.mostrar(pos);
            }
        });
    }
}
```

```
class SelectorFragment : Fragment() {
    val lugares by lazy { (activity!!.application as Aplicacion).lugares }
    val adaptador by lazy { (activity!!.application as Aplicacion).adaptador}
    val usoLugar by lazy { CasosUsoLugar(activity!!, lugares) }
    lateinit var recyclerView: RecyclerView

    override fun onCreateView(inflador: LayoutInflater, contenedor:
                        ViewGroup?, savedInstanceState: Bundle? ): View? {
        val vista =
                inflador.inflate(R.layout.fragment_selector,contenedor,false)
        recyclerView = vista.findViewById(R.id.recyclerView)
        return vista
    }

    override fun onActivityCreated(state: Bundle?) {
        super.onActivityCreated(state)
        recyclerView.apply {
            setHasFixedSize(true)
            layoutManager = LinearLayoutManager(context)
            adapter = adaptador
        }
        adaptador.onClick = {
            val pos = it.tag as Int
            usoLugar.mostrar(pos)
        }
    }
}
```

NOTA: *Tras incluir nuevas clases tendrás que indicar los imports adecuados. Para que Android Studio lo haga automáticamente pulsa* ***Alt-Intro****. La clase Fragment aparece en dos paquetes, por lo que te pedirá que selecciones uno de los dos. Utiliza el de la libreria androidx.fragment.app.Fragment.*

- android.app.Fragment
- android.support.v4.app.Fragment

El código de esta clase es similar al que teníamos antes en `MainActivity`, salvo que ahora extendemos a `Fragment` en vez de a `AppCompatActivity` y los métodos del ciclo de vida son diferentes.

Al igual que en una actividad, un *fragment* también tiene una vista asociada. En la actividad asociábamos la vista en el método `onCreate()`, llamando a `setContentView()`. En un *fragment* también disponemos del método `onCreate()`, pero no es aquí donde hay que asociar la vista. Se ha creado un nuevo método en el ciclo de vida, `onCreateView()`, con la finalidad de asociar su vista. En este método se nos pasan tres parámetros: un `LayoutInflater` que nos permite crear una vista a partir de un *layout* XML, el contenedor donde se insertará el *fragment* (en el punto siguiente veremos que se trata de un `LinearLayout`) y posibles valores guardados de una instancia anterior[53]. El método `onCreateView()` ha de devolver la vista ya creada. El hecho de disponer de este método resulta muy interesante, dado que nos permite cambiar la vista de un *fragment* sin tener que volverlo a crear.

Por otra parte, el método `onActivityCreated()` se llama cuando la actividad que contiene el *fragment* termina de crearse. Aprovecharemos este método para realizar tareas de inicialización como, por ejemplo, crear el adaptador y asociarlo al `RecyclerView`. Observa como la forma de trabajar con un `RecyclerView` es diferente cuando lo hacemos desde una actividad que cuando lo hacemos desde un `Fragment`. Aunque ha de quedar claro que en el fondo se realiza la misma tarea. En el siguiente esquema se compara cómo asociar el *layout* y el `RecyclerView` tanto en una actividad como en un *fragment*.

```
public class … extends Activity {

    @Override
    protected void onCreate(…) {
        …

setContentView(R.layout.layout);
        recyclerView = (RecyclerView)…
        recyclerView.setLayoutManager(…
        adaptador = new …;
    }
}
```

```
public class … extends Fragment {

  @Override
  public View onCreateView(…) {
     View vista = inflador.inflate(
                    R.layout.layout, …);
     recyclerView = (RecyclerView)…
     return vista;
  }

  @Override
  public void onActivityCreated(…) {
     super.onActivityCreated(…);
     recyclerView.setLayoutManager(…
     adaptador = new …;
  }
}
```

4. La actividad `MainActivity` va a visualizar el *layout content_main.xml*. Reemplaza su contenido por el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_width="match_parent"
```

53 http://www.youtube.com/watch?v=NMAJfqDOpBQ

```
    android:layout_height="match_parent"
    android:orientation="horizontal"
    app:layout_behavior="@string/appbar_scrolling_view_behavior">
    <fragment
       android:id="@+id/selector_fragment"
       android:name= "com.example.mislugares.presentacion.SelectorFragment"
       android:layout_width="0dp"
       android:layout_height="match_parent"
       android:layout_weight="1" />
</LinearLayout>
```

Como puedes ver, introducir el *fragment* desde un XML es muy sencillo. Simplemente añadimos una etiqueta `<fragment>` y en el atributo `name` indicamos el nombre de la clase del *fragment*. Este *fragment* es introducido en un `LinearLayout` que actuará de contenedor. Es habitual usar un contenedor para poder añadir nuevos *fragments* o reemplazarlos. Es importante incluir el atributo `layout_behavior` para su correcto funcionamiento dentro del `CoordinatorLayout`.

5. Elimina en `MainActivity` la declaración de la variable global `recyclerView`. En el método `onCreate()` elimina las líneas que inicializan `recyclerView` y la asignación del evento `onClick` para `adaptador`.

6. Ejecuta el proyecto. Verifica que la aplicación tiene la misma funcionalidad que antes.

   NOTA: *Puede parecer que no hemos conseguido gran cosa, dado que al final el funcionamiento es idéntico. Aunque resulta más complejo trabajar con fragments, Google recomienda que siempre diseñemos los elementos del IU basados en fragments, en lugar de en actividades. De esta forma, tendremos la posibilidad de mostrar varios elementos del IU a la vez en pantalla.*



Ejercicio: *Implementando un segundo fragment*

Recordemos que la aplicación que queremos hacer tiene que mostrar una serie de lugares, y al pulsar sobre uno de ellos tiene que mostrarnos información detallada sobre él. En este ejercicio crearemos un segundo *fragment* para mostrar la información del lugar, utilizando como base la actividad `VistaLugarActivity`. Haremos también que `MainActivity` muestre simultáneamente los dos *fragments* que hemos creado.

1. Desde el explorador del proyecto copia la actividad `VistaLugarActivity` (*Ctrl-C*) y realiza una copia (*Ctrl-V*) que se llame `VistaLugarFragment`.

2. Haz que la nueva clase herede de `Fragment` en lugar de `AppCompatActivity`. ***NOTA:*** *Importa esta clase del paquete androidx.fragment.app.Fragment.*

3. Elimina el método `onCreate()` y distribuye su código entre los siguientes:

```
@Override public View onCreateView(LayoutInflater inflador,
                         ViewGroup contenedor,Bundle savedInstanceState) {
```

```
    setHasOptionsMenu(true);
    View vista = inflador.inflate(R.layout.vista_lugar,contenedor,false);
    return vista;
}

@Override public void onActivityCreated(Bundle state) {
    super.onActivityCreated(state);

    lugares = ((Aplicacion) getActivity().getApplication()).lugares;
    adaptador = ((Aplicacion) getActivity().getApplication()).adaptador;
    usoLugar = new CasosUsoLugar(getActivity(), lugares);
    v = getView();
    foto = v.findViewById(R.id.foto);
    Bundle extras = getActivity().getIntent().getExtras();
    if (extras != null)
        pos = extras.getInt("pos", 0);
    else pos = 0;
    actualizaVistas();
}
```

```
val lugares by lazy { (activity!!.application as Aplicacion).lugares }
val adaptador by lazy { (activity!!.application as Aplicacion).adaptador }
val usoLugar by lazy { CasosUsoLugar(this activity!!, lugares)}
…

override fun onCreateView(inflador: LayoutInflater, contenedor:ViewGroup?,
                          savedInstanceState: Bundle? ): View? {
    setHasOptionsMenu(true)
    val vista = inflador.inflate(R.layout.vista_lugar, contenedor, false)
    return vista
}

override fun onActivityCreated(state: Bundle?) {
    super.onActivityCreated(state)
    pos = activity?.intent?.extras?.getInt("pos", 0) ?: 0
    actualizaVistas()
}
```

El *layout* que visualizará este *fragment* es el mismo que usábamos en la actividad, pero ahora se asigna en el método `onCreateView()`. La recogida de parámetros y la inicialización se realiza en `onActivityCreated()`.

4. Añade al principio de `actualizaVistas()`:

```
lugar = adaptador.lugarPosicion(pos);
```

Hemos hecho público este método para permitir que se llame desde fuera de la clase. Si el valor de `pos` ha cambiado tenemos que obtener el `lugar` adecuado.

5. En Java reemplaza todas las apariciones de `findViewById()` por `v.findViewById()`. Este método no está en la clase `Fragment`, pero sí que está en la clase `View`. Añade la variable global `v` de tipo `View`.

6. En Java, cambia el modificador de `onActivityResult()` de `protected` a `public`. Para poder sobreescribir un método es imprescindible que uses los mismos modificadores y, para el método en cuestión, son diferentes en la clase `Activity` que en `Fragment`.

7. Reemplaza el método `onCreateOptionsMenu()` por el siguiente:

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    inflater.inflate(R.menu.vista_lugar, menu);
    super.onCreateOptionsMenu(menu, inflater);
}
```

```
override fun onCreateOptionsMenu(menu: Menu, inflater: MenuInflater) {
   inflater.inflate(R.menu.vista_lugar, menu)
   super.onCreateOptionsMenu(menu, inflater)
}
```

Desde un *fragment* también podemos añadir ítems de menú a la actividad. El procedimiento es muy parecido, solo cambia el perfil del método.

8. Ya no estamos en una actividad has de reemplazar las apariciones de `this` por una referencia a la actividad del fragment:

```
Toast.makeText(this getActivity(), "…
```

```
Toast.makeText(this activity, "…
```

También:

```
getActivity().finish();
```

```
activity.finish()
```

9. Modifica *content_main.xml* para que muestre ambos *fragments*. Para ello añade el siguiente elemento al final del `LinearLayout`:

```
<fragment
     android:id="@+id/vista_lugar_fragment"
     android:name="com.example.mislugares.presentacion.VistaLugarFragment"
     android:layout_width="0dp"
     android:layout_height="match_parent"
     android:layout_weight="1" />
```

10. Abre la clase `MainActivity` y al final del método `onOptionsItemSelected()` asegúrate que el valor devuelto de la siguiente forma:

```
return super.onOptionsItemSelected(item);
```

De esta manera permitimos que el sistema pregunte a los *fragments* si tienen que procesar la selección de un ítem de menú.

11. Ejecuta la aplicación. Podrás ver como se muestran los dos *fragments* uno al lado del otro. De momento, el *fragment* de la derecha no muestra información de ningún lugar concreto.



Ejercicio: *Modificar el contenido de un fragment desde otro*

La aplicación creada hasta ahora no funciona como esperamos. En este ejercicio vamos a conseguir que, cuando se pulse sobre un elemento de la lista, el *fragment* de la derecha visualice la información del lugar seleccionado.

1. Reemplaza en `CasosUsoLugar` el método `mostrar()` por:

```java
public void mostrar(int pos) {
   VistaLugarFragment fragmentVista = obtenerFragmentVista();
   if (fragmentVista != null) {
      fragmentVista.pos = pos;
      fragmentVista._id = lugares.getAdaptador().idPosicion(pos);
      fragmentVista.actualizaVistas();
   } else {
      Intent intent = new Intent(actividad, VistaLugarActivity.class);
      intent.putExtra("pos", pos);
      actividad.startActivityForResult(intent, 0);
   }
}

public VistaLugarFragment obtenerFragmentVista()  {
   FragmentManager manejador = actividad.getSupportFragmentManager();
   return  (VistaLugarFragment)
           manejador.findFragmentById(R.id.vista_lugar_fragment);
}
```

```kotlin
fun mostrar(pos: Int) {
   var fragmentVista  = obtenerFragmentVista()
   if (fragmentVista != null) {
      fragmentVista.pos = pos
      fragmentVista._id = lugares.adaptador.idPosicion(pos)
      fragmentVista.actualizaVistas()
   } else {
      val i = Intent(actividad, VistaLugarActivity::class.java)
      i.putExtra("pos", pos);
      actividad.startActivity(i);
   }
}

fun obtenerFragmentVista(): VistaLugarFragment? {
   val manejador = actividad.supportFragmentManager
   return manejador.findFragmentById(R.id.vista_lugar_fragment) as
                                                   VistaLugarFragment?
}
```

Este método ha de visualizar el lugar solicitado. Comenzamos obteniendo una referencia al *fragment* con *id* `vista_lugar_fragment`. Si existe este *fragment* quiere decir que está ahora en pantalla y no es necesario crear una nueva actividad. Simplemente cambiando `pos` e `_id`, y llamando al método

actualizaVistas() conseguimos que se muestre la información en el *fragment* ya existente. En caso de que este *fragment* no exista (esto podrá pasar tras hacer uno de los próximos ejercicios), creamos una nueva actividad para mostrar la información.

2. En Java, para poder acceder a la propiedad pos e _id de VistaLugarFragment, cambia el modificador private por public.

3. Observa como aparece un error al tratar de obtener supportFragmentManager. Estamos trabajando con una variable de tipo Activity, pero este método solo está disponible en su descendiente FragmentActivity. Para resolverlo cambia el tipo de la propiedad.

```
public class CasosUsoLugar {
   private FragmentActivity actividad;
   …
   public CasosUsoLugar(FragmentActivity actividad, LugaresBD lugares …
```

```
class CasosUsoLugar(val actividad: FragmentActivity,
                    val lugares: LugaresBD, …
```

Este cambio implica que ya solo podremos usar estos casos de uso desde una actividad de la clase FragmentActivity. Nosotros lo estábamos haciendo desde AppCompactActivity. No vamos a tener problemas al tratarse de un descendiente de FragmentActivity.

java.lang.Object
↳ android.content.Context
↳ android.content.ContextWrapper
↳ android.view.ContextThemeWrapper
↳ android.app.Activity
↳ android.support.v4.app.FragmentActivity
↳ android.support.v7.app.AppCompatActivity

4. Ejecuta la aplicación y verifica que puedes cambiar el *fragment* de la derecha.

5. Si vas a preferencias y cambias el criterio de ordenación y acto seguido modificas la valoración del lugar en pantalla. Es posible que este se duplique en la lista de la izquierda. El problema se debe a que las variables pos y _id de VistaLugarFragment no tenían el valor correcto tras alterar el orden de la lista.

6. Para arreglarlo añade en MainActivity dentro de onActivityResult():

```
if (requestCode == RESULTADO_PREFERENCIAS) {
   adaptador.cursor = lugares.extraeCursor();
   adaptador.notifyDataSetChanged();
   if (usoLugar.obtenerFragmentVista() != null)
      usoLugar.mostrar(0);
}
```

En Kotlin el uso de ; es opcional. Lo que hacemos es averiguar si estamos visualizando dos *fragments* y en ese caso mostraremos en el *fragment* de la derecha el primer lugar de la lista. En caso contrario, solo se visualiza la lista y no existe VistaLugarFragment.



Ejercicio: *Adaptar CasosUsoLugar a fragments*

La clase `CasosUsoLugar` estaba pensada para ser usada desde una actividad. De hecho, era uno de los parámetros que se pasaban en el constructor. Gracias a este parámetro no solo se extraía el contexto, si no que también se usaba para invocar a `startActivityForResult()` para arrancar nuevas actividades y que luego se devuelva información a la actividad adecuada.

Pero ahora la situación ha cambiado, estos casos de uso no solo pueden ser utilizados por actividades, sino también por fragments. En este ejercicio vamos a introducir los cambios necesarios para que la respuesta de `startActivityForResult()` sea recogida por la actividad o fragment que está utilizando la clase.

1. Añade el siguiente atributo en `CasosUsoLugar`:

```
public class CasosUsoLugar {
   protected Fragment fragment;
   …
   public CasosUsoLugar(FragmentActivity actividad, Fragment fragment,
                        LugaresBD lugares, AdaptadorLugaresBD adaptador) {
      this.fragment = fragment;
      …
   }
```

```
open class CasosUsoLugar(
   open val actividad: FragmentActivity,
   open val fragment: Fragment?,
   open val lugares: LugaresBD,
   open val adaptador: AdaptadorLugaresBD)
```

La idea es que cuando se use desde una actividad el parámetro `fragment` se pase como `null`, y si es desde un fragment se pasarán tanto el parámetro `actividad` como `fragment`.

2. Reemplaza, todas las apariciones de `actividad.startActivityForResult` por:

```
if (fragment != null) fragment.startActivityForResult(…);
else                  actividad.startActivityForResult(…);
```

```
fragment?.startActivityForResult(…)
       ?:actividad.startActivityForResult(…)
```

Si nos han indicado un fragment llamamos desde este para que nos devuelva el resultado a este. En caso contrario lo hacemos desde la actividad.

3. En `MainActivity`, `VistaLugarActivity` y `EdicionLugarActivity` añade como nuevo parámetro `null`:

```
usoLugar = new CasosUsoLugar(this, null, lugares, adaptador);
```

```
val usoLugar by lazy { CasosUsoLugar(this, null, lugares, adaptador) }
```

4. En `VistaLugarFragment` y `SelectorFragment` añade como nuevo parámetro `this`:

```
usoLugar = new CasosUsoLugar(getActivity(), this, lugares, adaptador);
```

```
val usoLugar by lazy {CasosUsoLugar(activity!!, this, lugares, adaptador)}
```

5. Para que `onActivityResult()` se llame en la actividad y en los fragmentos has de llamar al `super` en `MainActivity`, al principio del método:

```
super.onActivityResult(requestCode, resultCode, data);
```



Ejercicio: *Introducir escuchador manualmente en el fragment*

Cuando definimos el *layout vista_lugar.xml* utilizamos el atributo `onClick` en varias vistas para asociar métodos que se ejecutan al pulsar sobre la vista. El problema es que estos métodos solo pueden definirse en una actividad y no en un *fragment*. Cuando diseñamos un *fragment* hemos de conseguir que sea reutilizable, por lo que todo su comportamiento debe definirse en la clase del *fragment*. Para resolver este problema, vamos a programar los escuchadores manualmente, en lugar de utilizar el atributo `onClick`. Más información sobre `onClick` y escuchadores de eventos en: http://youtu.be/OiVePqBmpcQ.

1. Abre el *layout vista_lugar.xml* y localiza el siguiente fragmento de código. Elimina la línea tachada y asegúrate que coincida el id:

```
<LinearLayout
     android:id="@+id/barra_url"
     android:layout_width="match_parent"
     android:layout_height="wrap_content"
     android:onClick="verPgWeb"
     android:orientation="horizontal" >
```

2. Abre la clase `VistaLugarFragment` y añade en el método `onActivityCreated()` el siguiente código. **En Java** tras obtener `v`:

```
v.findViewById(R.id.barra_url).setOnClickListener(new OnClickListener () {
   public void onClick(View view) {  usoLugar.verPgWeb(lugar); } });
```

```
barra_url.setOnClickListener { usoLugar.verPgWeb(lugar) }
```

***NOTA:** Cuando pulses **Alt-Intro** para incluir los imports de las nuevas clases, selecciona el paquete marcado:*

android.view.View.OnClickListener
android.content.DialogInterface.OnClickListener

3. Ejecuta la aplicación y verifica que, al pulsar en la vista de un lugar, sobre la URL o su icono se abre la página web correspondiente.

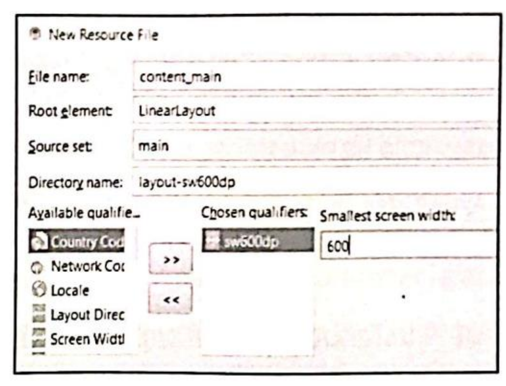
4. Repite esta operación para todas las vistas del *layout* donde se haya utilizado el atributo `onClick`.



Ejercicio: *Mostrar dos fragments solo con pantallas grandes*

Cuando ejecutamos la aplicación en una pantalla pequeña, como la de un teléfono, no tiene ningún sentido mostrar dos *fragments* simultáneamente. Esto solo nos interesa en una tableta. Para conseguir este doble funcionamiento vamos a trabajar con dos *layouts* diferentes. Cargaremos uno u otro aprovechando los recursos alternativos de Android.

1. En el explorador del proyecto, pulsa con el botón derecho sobre la carpeta *res/layout.* y selecciona *New > Layout Resource File*. En *File name:* introduce *content_main*; en *Available qualifiers:* selecciona *Smallest Screen Width*; y el valor *600*:



   Se creará la carpeta *layout-sw600dp*. Los recursos de esta carpeta se cargarán cuando la aplicación se ejecute en una pantalla de 7' o más[54].

2. Realiza una copia del contenido de *content_main.xml* por defecto al nuevo recurso que acabas de crear.
3. Elimina en el *content_main.xml* por defecto el segundo de los dos *fragments* que contiene.
4. Ejecuta la aplicación en un dispositivo de pantalla pequeña y en uno de más de 7'. Observa como se muestra uno o dos *fragments* según el tamaño de la pantalla.



Práctica: *Simplificación de la actividad VistaLugarActivity*

Si comparas el código de la actividad `VistaLugarActivity` con el de `VistaLugarFragment` verás que son casi idénticos. Dejar el mismo código en dos clases diferentes es un grave error de programación. Trata de modificar la actividad `VistaLugarActivity` para que se limite a visualizar el *fragment* `VistaLugarFragment` en su interior.

[54] http://www.androidcurso.com/images/dcomg/ficheros/recursos_alternativos.pdf



Solución:

Crea el layout `activity_vista_lugar.xml`:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="horizontal" >
   <fragment
     android:id="@+id/vista_lugar_fragment"
     android:name="com.example.mislugares.presentacion.VistaLugarFragment"
     android:layout_width="0dp"
     android:layout_height="match_parent"
     android:layout_weight="1"
     android:clickable="true"/>
</LinearLayout>
```

En `VistaLugarActivity`: solo ha de estar el método `onCreate()` que cargue el layout:

```
setContentView(R.layout.activity_vista_lugar);
```



Ejercicio: *Ajustando comportamiento al borrar un lugar con fragments*

Dentro de la clase `CasosUsoLugar`, el código para borrar un lugar termina con:

```
actividad.finish();
```

Si trabajamos solo en una actividad, el funcionamiento es correcto. Tras borrar el lugar cerramos la actividad dado que no tiene sentido mostrar un lugar que ya no existe. Pero si trabajamos con dos *fragments* visualizándose juntos, al cerrar la actividad se cerrarán los dos y saldremos de la aplicación. En el presente ejercicio corregiremos este comportamiento no deseado.

1. En el método `borrar()` reemplaza `actividad.finish()` por el código siguiente:

```
if (obtenerFragmentSelector() == null) {
   actividad.finish();
} else {

   mostrar(0);
}
```

```
if (obtenerFragmentSelector() == null) {
   actividad.finish()
} else {
   mostrar(0)
}
```

Tras borrar el lugar trataremos de averiguar si estamos en una configuración con dos *fragments*. Esto ocurrirá cuando podemos obtener una referencia de `selector_fragment`. Si no existe, realizamos la misma acción de antes. Si existe, hacer que se muestre un lugar con código diferente del borrado, en este caso se muestra el primero de la lista.

2. Añade la siguiente función:

```
private SelectorFragment obtenerFragmentSelector() {
   FragmentManager manejador = actividad.getSupportFragmentManager();
   return
(SelectorFragment)manejador.findFragmentById(R.id.selector_fragment);
}
```

```
fun obtenerFragmentSelector(): SelectorFragment? {
   val manejador = actividad.supportFragmentManager
   return manejador.findFragmentById(R.id.selector_fragment) as
                                                    SelectorFragment?
}
```

3. Ejecuta la aplicación y verifica el resultado.
4. Al borrar todos los lugares de la lista, trabajando con una tableta, verás que se produce un error. Ni siquiera podrás volver a arrancar la aplicación.
5. Para resolverlo añade en `VistaLugarFragment` al comienzo de `actualizaVistas`:

```
if (lugares.tamaño() == 0) return;
```

6. Ejecuta la aplicación y verifica el resultado.



Preguntas de repaso: *Fragments*

ANEXO C.

Referencia Java

Variables

Constantes `final`

```
final <tipo> <CONSTANTE> = <valor>;
```

Donde en <valor> se escribe: byte:`(byte)64`, short:`(short)64`, int: `64`, long: `64L`, float: `64.0f`, double: `64.0`, char: `'@'` o `'\u0040'`, boolean: `true` / `false` objetos: `null`, String: "64", vectores: `{<valor>, <valor>, …}`

Ejemplo: `final int MAX_ELEM = 20;`

Tipos simples o primitivos

```
<tipo_simple> <variable> [= <valor>];
```

Ejemplo: `int i = 0;`

tipo	tamaño	rango		envolvente
byte	8 bits	-128	127	Byte
short	16 bits	-32.768	32.767	Short
int	32 bits	-2.147.483.648	2.147.483.647	Integer
long	64 bits	$-9.223.372 \cdot 10^{12}$	9.223.372.036.854.775.807	Long
float	32 bits	$-3.4 \cdot 10^{38}$	$3.4 \cdot 10^{38}$ (mínimo $1.4 \cdot 10^{-45}$)	Float
double	64 bits	$-1.8 \cdot 10^{308}$	$1.8 \cdot 10^{308}$(mínimo $4.9 \cdot 10^{-324}$)	Double
boolean		false	true	Boolean
char	16 bits	Unicode 0	Unicode $2^{16}-1$	Character
void	0 bits	-	-	Void

Tipo compuesto `new`

```
<tipo_compuesto> <variable> = new <tipo_compuesto>(<param>);
```

Pueden ser: *arrays* o clases. Dentro de las clases existen algunas especiales: *envolventes*, `String`, *colecciones* y *enumerados*.

Siempre son referencias (punteros)

Arrays

YouTube

YouTube

```
<tipo><array>[]..[] = new <tipo>[<num>]..[<num>];
```

El primer elemento es el 0, al crearlos (*new*) hay que saber su tamaño.

```
float v[] = new float[10]; //Una dimensión y 10 elementos
float M[][]= new float[3][4]; //Dos dimensiones
String s[] = {"hola", "adios"}; // elementos inicializados
```

```
for (int i = 0; i < M.length; i++)
  for (int j = 0; j < M[i].length; j++)
    M[i][j] = 0;
```

Envolvente (wrapper) YouTube

Para cada uno de los tipos simples existe una clase equivalente, con constantes y métodos que nos pueden ser útiles. Véase tabla en *variable simple*. Véase *conversión de tipos*.

Cadena caracteres `String`

```
String <nombre_variable> [= "<cadena de caracteres>"];
```

Ejemplo: `String s = "Hola";` o `String s = new String("Hola");`

Métodos de la clase `String`:

```
.equals(String s2) //compara dos Strings
.clone() //crea una copia de un String
.charAt(int pos) //retorna el carácter en una posición
.concat(String s2) //concatena con otro String
.indexOf(char c, int pos) //devuelve posición de un carácter
.length() //devuelve la longitud del String
.replace(char c1, char c2) // reemplaza un carácter por otro
.substring(int pos1, int pos2) // extrae una porción
.toLowerCase() // convierte el String a minúsculas
.toUpperCase() // convierte el String a mayúsculas
.valueOf(int/float/... numero) //convierte un número a String
```

Colecciones poli[Media]

La API de Java nos proporciona colecciones donde guardar series de objetos de cualquier clase. Dichas colecciones no forman parte del lenguaje, si no que son clases definidas en el paquete *java.util*.

```
<Tipo_colecc><<Clase>> <colección> =
              new <Tipo_colecc><<Clase>>();
```

Hay tres tipos, cada uno con una interfaz común y diferentes implementaciones:

Listas – interfaz: `List<E>` Estructura secuencial, donde cada elemento tiene un índice o posición: `ArrayList<E>` (acceso rápido), `LinkedList<E>` (inserción/borrado rápido), `Stack<E>` (pila), `Vector<E>` (obsoleto).

Conjunto – interfaz: `Set<E>` Los elementos no tienen un orden y no se permiten duplicados: `HashSet<E>` (la implementación usa tabla *hash*), `LinkedHashSet<E>` (+ doble lista enlazada), `TreeSet<E>` (usa árbol).

Diccionario – interfaz: `Map<K,V>` Cada elemento tiene asociada una clave que usaremos para recuperarlo (en lugar del índice de un vector): `HashMap<K,V>`, `TreeMap<K,V>`, `LinkedHashMap<K,V>`

Las interfaces `Iterator` y `ListIterator` facilitan recorrer colecciones. La clase estática `Collections` nos ofrece herramientas para ordenar y buscar en colecciones.

```
ArrayList<Complejo> lista = new ArrayList<Complejo>();
lista.add( new Complejo(1.0, 5.0) );
lista.add( new Complejo(2.0, 4.2) );
lista.add( new Complejo(3.0, 0.0) );
```

<table>
<tr><td></td><td><pre>for (Complejo c: lista) {
   System.out.println(c);
}</pre></td></tr>
<tr><td>Enumerado
enum
(Java 5)[55]
poli[Media]</td><td><pre>enum <nombre_enumeracion> { <CONSTANTE>, ..., < CONSTANTE> }</pre>Ejemplo: <pre>enum estacion {PRIMAVERA, VERANO, OTOÑO, INVIERNO};
        estacion a = estacion. VERANO;</pre></td></tr>
<tr><td>Ámbito</td><td>Indica la vida de una variable, se determina por la ubicación de llaves { } donde se ha definido.<pre>{
  int a = 10; // solo a disponible
  {
    int b = 20; // a y b disponibles
  }
  // solo a disponible
}</pre></td></tr>
</table>

Expresiones y sentencias

<table>
<tr><td>Comentarios</td><td colspan="2"><pre>// Comentario de una línea
/* Comentario de
   varias líneas */
/** Comentario javadoc: para crear automáticamente la
    documentación de tu clase */</pre></td></tr>
<tr><td>Operadores
YouTube</td><td colspan="2">asignación: <code>=</code>
aritméticos: <code>++, --, +, -, *, /, %</code>
comparación: <code>==, !=, <, <=, >, >=, !, &&, ||, ?:</code>
manejo bits: <code>&, |, ^, ~, <<, >>, >>></code>
conversión: <code>(<tipo>)</code></td></tr>
<tr><td>Conversión de tipos
YouTube</td><td colspan="2">Entre tipos compatibles se puede hacer asignación directa o utilizar un typecast.<pre>byte b = 3;
int i = b; float f = i; //int a byte y float a int
b =(byte)i; // hay que hacer un typecast
String s = Integer.toString(i);
b = Byte.parseByte(s);</pre></td></tr>
<tr><td>Estructura condicional
if
else
switch</td><td><pre>if (<condición>) {
  <instrucciones>;
} else {
  <instrucciones>;
}</pre></td><td><pre>if (b != 0) {
   System.out.println(
     "x= "+a/b);
} else {
   System.out.println(
          "Error");
}</pre></td></tr>
</table>

[55] Solo disponible a partir de la versión 5 de Java.

case **default** YouTube	`switch (<expresión>) {` `  case <valor>:` `    <instrucciones>;` `    [break;]` `  case <valor>:` `    <instrucciones>;` `    [break;]` `  ...` `  [default:` `    <instrucciones>;]` `}`	`switch (opcion) {` `  case 1:` `    x = x * Math.sqrt(y);` `    break;` `  case 2:` `  case 3:` `    x = x / Math.log(y);` `    break;` `  default:` `  System.out.println(` `            "Error");` `}`
Estructuras iterativas **while** **do** **for** YouTube	`while (<condición>) {` `  <instrucciones>;` `}`	`i = 0;` `while (i < 10) {` `   v[i]=0;` `   i++;` `}`
	`do {` `  <instrucciones>;` `} while (<condición>)`	`i = 0;` `do {` `   v[i]=0;` `   i++;` `} while (i < 10)`
	`for (<inicialización>;` `   <comparación>; <incremento>) {` `  <instrucciones>;` `}`	`for (i = 0; i < 10; i++){` `   v[i]=0;` `}`
	`for (<tipo> <variable>` `                 :<colección>) {` `  <instrucciones>;` `}           //(Java 5)`	`for (Complejo c: lista){` `   c.toString();` `}`
Sentencias de salto **break** **continue** **return** **exit**	`break;` Fuerza la terminación inmediata de un bucle o de un *switch*. `continue;` Fuerza una nueva iteración del bucle o salta a una etiqueta. `return [<valor>]:` Sale de la función, puede devolver un valor. `exit([int código]);` Sale del programa, puede devolver un código.	

Clases y objetos

Clases **class** poli[Media][56]	Cada clase ha de estar en un fichero separado con el mismo nombre de la clase y con extensión `.class`. Por convenio, los identificadores de clase se escriben en mayúscula. `class <Clase> [extends <Clase_padre>][implement <interfaces>] {` `  //declaración de atributos`

[56] Puedes encontrar un vídeo explicactivo en www.androidcurso.com

```
  [visibilidad] [modificadores] <tipo> <atributo> [= valor];
  ...
  //declaración de constructor
  public <Clasee>(<argumentos>) {
    <instrucciones>;
  }
  //declaración de métodos
  [visibilidad] [modificadores] <tipo>
                                 <método>(<argumentos>) {
    <instrucciones>;
  }
  ...
}
```

donde: `[visibilidad] = public, protected o privateB`
`[modificadores] = final, static y abstract`

Ejemplo:

```
class Complejo {
  private double re, im;
  public Complejo(double re, double im) {
   this.re = re; this.im = im;
  }
  public String toString() {
   return(new String(re + "+" + im + "i"));
  }
  public void suma(Complejo v) {
   re = re + v.re;
   im = im + v.im;
  }
}
```

Uso de objetos:

```
Complejo z, w;
z = new Complejo(-1.5, 3.0);
w = new Complejo(-1.2, 2.4);
z.suma(w);
System.out.println("Complejo: " + z.toString());
```

Sobrecarga

YouTube

Podemos escribir dos métodos con el mismo nombre si cambian sus parámetros.

```
public <tipo> <método>(<parámetros>) {
    <instrucciones>;
}
public <tipo> <método>(<otros parámetros>) {
    <otras instrucciones>;
}
```

Ejemplo:

```
public Complejo sumar(Complejo c) {
```

```
    return new Complejo(re + c.re, im + c.im);
  }
  public Complejo sumar(double r, double i) {
    return new Complejo(re + r, im + i);
  }
```

Herencia

`extends` `@Override` `super.` poli[Media]

```
class <Clase_hija> extends <Clase_padre> {
  ...
}
```

La clase hija heredará los atributos y métodos de la clase padre. Un objeto de la clase hija también lo es de la clase padre y de todos sus antecesores. La clase hija puede volver a definir los métodos de la clase padre, en cuyo caso es recomendable (no obligatorio) indicarlo con `@Override`; de esta forma evitamos errores habituales cuando cambiamos algún carácter o parámetro. Si un método ha sido sobrescrito, podemos acceder al de la clase padre con el prefijo `super.<método>(<parámetros>)`. Véase ejemplo del siguiente apartado.

Constructor

`super()` YouTube

Método que se ejecuta automáticamente cuando se instancia un objeto de una clase. Ha de tener el mismo nombre que la clase. Cuando se crea un objeto, todos sus atributos se inicializan en memoria a cero y las referencias serán `null`. Una clase puede tener más de un constructor (véase sobrecarga). Un constructor suele comenzar llamando al constructor de la clase padre, para lo cual escribiremos como primera línea de código: `super(<parámetros>);`

```
class ComplejoAmpliado extends Complejo {
  private Boolean esReal;
  public ComplejoAmpliado(double re, double im) {
    super(re, im);
    esReal = im ==0;
  }
  public ComplejoAmpliado(double re) {
    super(re, 0);
    esReal = true;
  }
  @Override
  public Complejo sumar(double re, double im) {
    esReal = im == -this.im;
    return super.sumar(re,im);
  }
  public boolean esReal(){
    return esReal;
  }
}
```

Visibilidad

La visibilidad indica quién puede acceder a un atributo o métodos. Se define antecediendo una de las palabras (por defecto `public`).

public private protected
poli[Media]

`public`: Accesibles para cualquier clase.

`private`: Solo son accesibles para la clase actual.

`protected`: Solo para la clase que los ha declarado y para sus descendientes.

`<si no indicamos nada>`: Solo son accesibles para clases de nuestro paquete.

Modificadores
final abstract static
YouTube

`final`: Se utiliza para declarar una constante (delante de un atributo), un método que no se podrá redefinir (delante de un método) o una clase de la que ya no se podrá heredar (delante de una clase).

`abstract`: Denota un método del cual no se escribirá código. Las clases con métodos abstractos no se pueden instanciar. Las clases descendientes deberán escribir el código de sus métodos abstractos.

`static`: Se aplica a los atributos y métodos de una clase que pueden utilizarse sin crear un objeto que instancie dicha clase. El valor de un atributo estático, además, es compartido por todos los objetos de dicha clase.

Comparación y asignación de objetos
equals y ==
clone y =

Podemos comparar valores de variables con el operador ==, y asignar un valor a una variable con el operador =. En cambio, el nombre de un objeto de una clase no contiene los valores de los atributos, si no la posición de memoria donde residen dichos valores de los atributos (referencia indirecta). Utilizaremos el operador == para saber si dos objetos ocupan la misma posición de memoria (son el mismo objeto), mientras que utilizaremos el método `equals(<Obeto>)` para saber si sus atributos tienen los mismos valores. Utilizaremos el operador = para asignar a un objeto otro objeto que ya existe (serán el mismo objeto) y `clone()` para crear una copia idéntica en un nuevo objeto.

Polimorfismo
instanceof
poli[Media]

Se trata de declarar un objeto de una clase, pero instanciarlo como un descendiente de dicha clase (lo contrario no es posible):

```
<Clase_padre> <objeto> = new <Clase_hija>(<parámetros>);
```

Podemos preguntar si un objeto es de una determinada clase con:

```
<objeto> instanceof <Clase>
```

Podemos hacer un *tipecast* a un objeto para considerarlo de otra clase:

```
(<Clase>) <objeto>
```

Ejemplo:

```
Complejo c = new ComplejoAmpliado(12.4);
if (c instanceof Complejo)… //true
if (c instanceof ComplejoAmpliado)… //true
if (((ComplejoAmpliado)c).esReal())…
```

Recolector de basura
finalize()

Cuando termina el ámbito de un objeto (véase sección "Ámbito") y no existen más referencias a él, el sistema lo elimina automáticamente.

```
{
   Complejo a; // solo a disponible, pero no inicializado
   {
```

```
      Complejo b = new Complejo(1.5,1.0); // Se crea un objeto
      a = b; // Dos referencias a un mismo objeto
    }
    // solo a disponible
  } // el objeto es destruido liberando su memoria
```

Para eliminar un objeto, el sistema llama a su método `finalize()`. Podemos rescribir este método en nuestras clases:

```
@Override protected void finalize() throws Throwable {
  try {
    close();          // cerramos fichero
  } finally {
    super.finalize();
  }
}
```

Métodos con argumentos variables en número (Java 5)

```
[visibilidad] [modificadores] <tipo> <método>(<Clase>... args) {
  <instrucciones>;
}
```

Ejemplo:

```
public void sumar(Complejo... args) {
  for (int i = 0; i < args.length; i++) {
    re = re + args[i].re;
    im = im + args[i].im;
  }
}
```

Interfaces

`Interface`

YouTube

Clase completamente abstracta. No tiene atributos y ninguno de sus métodos tiene código (en Java no existe la herencia múltiple, pero una clase puede implementar una o más interfaces, adquiriendo sus tipos).

```
interface <interface> [extends <interface_padre>] {
  [visibilidad] [modificadores] <tipo> <metodo1>(<argumentos>);
  [visibilidad] [modificadores] <tipo> <metodo2>(<argumentos>);
  ...
}
```

Para heredar de una interfaz:

```
class <Nombre_clase> extends <clase_padre> implements
                    <interface1>, <interface2>, ... {
  ...
}
```

Otros

Paquetes

`package`
`import`

Los paquetes son una forma de organizar grupos de clases. Resuelven el problema del conflicto entre los nombres de las clases. Por ejemplo, la clase `Font` se ha creado en cientos de paquetes Java. Para referirnos a una de ellas es obligatorio indicar el paquete al que pertenece. Por ejemplo, `java.awt.Font`.

Al inicio del fichero de una clase se debe indicar su paquete:

```
package carpeta.subcarpeta.…;
```

Para usar una clase de otro paquete se indica su paquete:

```
java.awt.Font fuente=new java.awt.Font(…);
```

Para abreviar, tambien podemos importar la clase de un paquete:

```
import java.awt.Font;
```

y utilizar solo el nombre de la clase:

```
Font fuente=new Font(…);
```

Para importar todas las clases de un paquete:

```
import java.awt.*;
```

Excepciones

try
catch
finally

YouTube

poli[Media]

```
try {
   Código donde se pueden producir excepciones.
}
catch (TipoExcepcion1 nombreExcepcion) {
   Código a ejecutar si se produce una excepción TipoExcepcion1.
}
catch (Excepcion nombreExcepcion) {
   Código a ejecutar si se produce una excepción de cualquier tipo.
}
finally {
   Código a ejecutar tanto si se produce una excepción como si no.
}
```

Ejemplo:

```
String salario; BufferedReader fichero1;
try {
  fichero1 = new BufferedReader(new FileReader("\file"));
  salario = fichero1.readLine();
  salario = (new Integer(Integer.parseInt(salario)*10)
                                      .toString());
}
catch (IOException e) {
  System.err.println(e);
}
catch (NumberFormatException e) {
  System.err.println("No es un número");
}
catch (Exception e) {
  System.err.println("Excepción de cualquier tipo");
}
finally {
  fichero1.close();
}
```

Hilos de ejecución `Thread` YouTube	Creación de un nuevo hilo que llama una vez al método `hazTrabajo()`: `class MiHilo extends Thread {` `  @Override public void run() {` `    hazTrabajo();` `  }` `}` Para ejecutarlo: `MiHilo hilo = new MiHilo ();` `hilo.start();` Creación de un nuevo hilo que llama continuamente al método `hazTrabajo()` y que puede ser pausado y detenido: `class MiHilo extends Thread {` `  private boolean pausa,corriendo;` `  public synchronized void pausar() {` `    pausa = true;` `  }` `  public synchronized void reanudar() {` `    pausa = false;` `    notify();` `  }` `  public void detener() {` `    corriendo = false;` `    if (pausa) reanudar();` `  }` `  @Override public void run() {` `    corriendo = true;` `    while (corriendo) {` `      hazTrabajo();` `      synchronized (this) {` `        while (pausa) {` `          try {` `            wait();` `          } catch (Exception e) {}` `        }` `      }` `    }` `  }` `}`
Secciones críticas `Synchronized`	Cada vez que un hilo de ejecución va a entrar en un método o bloque de instrucciones marcado con `synchronized` se comprueba si ya hay otro hilo dentro de la sección crítica de este objeto (formada por todos los bloques de instrucciones marcados con `synchronized`). Si ya hay otro hilo dentro, entonces el hilo actual es suspendido y ha de esperar hasta que la sección

crítica quede libre. Para que un método pertenezca a la sección critica de objeto escribe:

```
public synchronized void metodo() {...}
```

o, para que un bloque pertenezca a la sección critica de objeto:

```
synchronized (this) {...}
```

Recuerda: la sección crítica se define a nivel de objeto, no de clase. Solo se define una sección crítica por objeto.

Para conseguir una sección critica por clase escribe:

```
public static synchronized void metodo() {...}
```

o `synchronized (MiClase.class) {...}`

Genericidad (Java 5) poli[Media]

Permite independizar el código del tipo de datos sobre el que se aplica.

```
Public class Caja<T> {
  private T dato;
  public void poner(T d) {dato = d;}
  public T sacar() {return dato;}
}
```

Inicio de la aplicación `main`

Una aplicación Java debe tener al menos una clase con un tipo de método denominado `main`. Será el primer método en ser ejecutado:

```
class <Clase> {
  public static void main(String[] main) {
    <instrucciones>; }
}
```

ANEXO D. Referencia de la clase View y sus descendientes

http://www.androidcurso.com/index.php/referencias/referencia-clase-view

ANEXO E. Sufijos utilizados en recursos alternativos

http://www.androidcurso.com/index.php/referencias/recursos-alternativos